METHODS FOR TEACHING DIVERSE ROBOT SKILLS: LEVERAGING PRIORS, GEOMETRY, AND DYNAMICS

A Dissertation
Presented to
The Academic Faculty

By

Muhammad Asif Rana

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering
College of Engineering

Georgia Institute of Technology

December 2020

METHODS FOR TEACHING DIVERSE ROBOT SKILLS: LEVERAGING PRIORS, GEOMETRY, AND DYNAMICS

Thesis committee:

Dr. Sonia Chernova, Advisor
School of Interactive Computing
Georgia Institute of Technology

Dr. Seth Hutchinson
School of Interactive Computing
Georgia Institute of Technology

Dr. Byron Boots
Paul G. Allen School of Computer Science and Engineering
University of Washington

Dr. Matthew Gombolay
School of Interactive Computing
Georgia Institute of Technology

Dr. Tucker Hermans
School of Computing
University of Utah

Date approved: September 4, 2020

I would rather have questions that can't be answered than answers that can't be questioned.

Richard P. Feynman

To my family and friends.

ACKNOWLEDGMENTS

The road to this junction has been long, curvy and full of hurdles. The journey till this point would not have been possible without the support of all the people that have stood by my side.

First and foremost, I would like to thank my advisor, Sonia Chernova. I still distinctly remember my first interaction with her. At that time, after multiple failed attempts at finding a compatible advisor, I was on the brink of leaving the PhD program. Sonia not only rescued me out of that situation, but from thereon also helped grow tremendously as a researcher. She gave me immense amount of freedom to carry out my research at my own terms, while also guided me whenever I got stuck. I feel myself really lucky to have her as my advisor. I would also like to extend my gratitude to Byron Boots. Throughout my time at Georgia Tech and at NVIDIA, Byron has helped me navigate my research in newer and interesting directions. Whenever Sonia ran out of ideas, Byron was always there to bounce new ideas off. I have gained a lot from Byron's insights in the area of machine learning and particularly imitation learning. In Sonia and Byron, I found true roles models, whom I wish to emulate going forward in my life.

I would like to thank all the rest of my committee members as well, including Seth Hutchinson, Matthew Gombolay, and Tucker Hermans. I am grateful for their valuable feedback and insights on my research. I also want to thank my mentors and colleagues at NVIDIA, especially Nathan Ratliff, Fabio Ramos, Karl Van Wyk, and Dieter Fox. I have thoroughly enjoyed my time at NVIDIA and learned a lot in terms of practical applications of my research work.

Next, I would also like to acknowledge all my collaborators who have helped me in my research work, namely Mustafa Mukadam, Anqi Li, Harish Ravichandar, Reza Ahmadzadeh, Daphne Chen, and Vivian Chu. I would not have been able to publish a single research paper without the support of the aforementioned people. I am also thankful for having

the company of my amazing labmates throughout this journey, specifically, David Kent, Siddhartha Banerjee, Lakshmi Nair, Angel Daruna, Weiyu Liu, Jonathan Balloch, Kalesha Bullard, and Tesca Fitzgerald. I will always fondly remember the fun lunch/coffee chats with my labmates. My friends outside lab, especially my little Atlanta family need to be acknowledged here too. I was lucky to have always been surrounded by people with whom I would forget about all the worries of life.

Last but not the least, I want to thank my family for always believing in me. I would especially acknowledge the support and efforts of my mother who pushed me to aim and achieve better and bigger goals in life. Whatever I am today, its because of her. I also want to thank my wife, who joined me in the final and probably the hardest part of my PhD journey.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Functioning in the real world requires robots to reason about and generate motions for execution of complex tasks, in potentially unstructured and dynamic environments. Early generations of robots were limited to simple tasks in controlled environments, where only a single skill was often required. To deal with the diversity of tasks and environments associated with the real world, robots should instead have access to a library of skills. Instead of pre-programming all the desired skills, a procedure which is cumbersome and often infeasible, it is beneficial to have a framework that allows robots to acquire new skills when required. One such framework is learning from demonstration, which provides a channel for robots to learn skills from everyday users. This dissertation provides methods for learning skills from human demonstrations.

Skill learning from human demonstrations carries certain challenges. Skills can be vastly different, enforcing a range of motion constraints. Human demonstrations are also often limited in number. Lastly, generalization of learned skills can be tied to generating motions that need to satisfy additional pre-specified constraints. These constraints can be associated with feasibility, requiring motions compliant with robot's kinematics and its environment, or they may be linked to coordination, requiring correlated motions of several robot body parts. To contend with the diversity of skills, the presence of feasibility and coordination constraints, and the scarcity of data, it is beneficial to impose structure in the skill representation. The structure incorporates domain knowledge in the representation, enabling desirable generalization even when access to large amounts data is hard.

The objective of this dissertation is to develop a family of techniques that allow robots to sample-efficiently learn diverse skills from human demonstrations, and subsequently *generalize* the skills to novel contexts while satisfying additional constraints that may exist, concerning the *feasibility* and *coordination* of robot motions. Each proposed method comes with a structured representation, suitable for tackling the challenges associated

with a subset of skills. Specifically, we present: *(i)* a structured multi-coordinate cost learning framework coupled with an optimization routine, that generalizes skills requiring preservation of multiple geometric properties of motions, *(ii)* a structured prior representation employed in a probabilistic inference framework, geared towards generating optimal and feasibility-constrained motions, *(iii)* a stable dynamical system representation, suitable for learning skills aimed at motions that can react instantly to dynamic perturbation, and *(iv)* a tree-structured stable dynamical system which synthesizes multiple dynamical system into one, and learns skills dictating feasible and coordinated, yet reactive robot motions. As a preliminary to the aforementioned learning techniques, this dissertation also provides an over-arching benchmarking effort to identify the key challenges associated with skill learning from demonstration.

CHAPTER 1
INTRODUCTION

Recent advances in technology have enabled robots to assume roles alongside humans in several fields, including healthcare, defense, education, search and rescue, and even exploratory missions in space and underwater. For a robot deployed in either of the aforementioned roles, it is often desirable to execute a wide range of tasks in potentially diverse environments. However, the tasks that a robot may encounter during its lifetime may not be known at deployment. Thus it is not possible to predict and pre-program the specifications for all the future tasks assigned to the robot. It is imperative for robots to instead have the capability to continually expand their skill sets in order to execute new tasks as they are encountered.

In most assistive or collaborative roles, the robot has access to a human with task knowledge. The robot can thus leverage the human knowledge whenever new task specifications are desired. Most early approaches towards task knowledge transfer from humans to robots, were limited to hand-coded simple repetitive tasks e.g. in factory settings. Hand-specifying complex tasks however is a cumbersome or sometimes infeasible process, especially for novice users. Learning from demonstration (LfD) [1, 2, 3] seeks to address this problem by enabling everyday users to transfer task specifications to a robot by way of demonstrations, instead of explicit programming.

A robotic task execution typically has two levels of abstraction: *(i)* high-level task planning, and *(ii)* low-level motion generation. At the task planning abstraction, a task is divided into several less complicated tasks, also called *skills*, or *movement primitives*, which are sequentially executed. For each skill, the motion generation abstraction further reasons about the motion constraints or subgoals (e.g. start/goal states) passed by the higher level abstraction, and generates a sequence of motor commands for the robot. Learning from demonstration has been successfully applied in literature to either of the aforementioned

levels of abstractions [4, 5, 6, 7, 8, 9, 10].

In this dissertation, we focus on learning robot manipulation skills from human demonstrations. More specifically, we learn skills, successful execution of which requires constrained motions of an articulated robot manipulator[1]. In a typical skill learning setting, a human teacher provides multiple demonstrations in different contexts, whereby each demonstration is a continuous motion or trajectory. From the demonstrations, motion constraints associated with the skill are extracted in terms of a mathematical representation, also called a skill model, which can be employed to autonomously reproduce the skill. There are multiple challenges associated with skill learning from human demonstrations, some of which are enlisted below.

1. Since it is not possible for a human teacher to provide demonstrations for every situation the robot might encounter, it is critical for a learned skill to ***generalize*** to new contexts (e.g. new robot start/goal states, and new environments). Providing demonstrations is a cumbersome process in general, thus skill learning approaches often have to make use of only a ***few*** demonstrations.
2. In the presence of robot's kinematic constraints, and/or environmental constraints brought about by the presence of obstacles, a skill model should be capable of generating motions that are ***feasible*** under these constraints.
3. Certain skills may impose motion constraints that live on task spaces spanning across several body parts of a robot. To comply with such skills, a skill model should be capable of generating simultaneous and ***coordinated*** motions of multiple robot body parts.

Although it is important to address the aforementioned challenges while devising a skill learning technique, it also important to appreciate that skills can be highly diverse in the type of motion constraints they encompass. Execution of certain skills may require a desired shape of motion needs to be maintained. On the other hand, some skills can require completion within a fixed time duration. Additionally, there also exist skills focused on

[1]We assume that a task planner or human expert provides task decomposition structure beforehand.

dynamic environments and hence require reactive motion generation. High dimensional representations, while capable of encoding diverse skills, are prone to overfitting. Hence, when data is scarce, such representations may not generalize a skill in a human desired way. Furthermore, feasible and/or coordinated motions are not guaranteed unless such constraints are embedded in the skill representation structure. Therefore instead of employing a one-for-all skill representation, it is beneficial to have several skill representations at our disposal, each with a restricted hypothesis class spanning a subset of skills. Such skill representations can enforce a priori known properties and constraints associated with a given skill, and thus enable ***sample-efficient*** skill learning.

This dissertation introduces a family of skill learning approaches, each with a novel structure in representation. We employ geometric features, structured priors, and stable[2] dynamical systems (DS) to represent skills. Unlike most existing methods, that are limited to encoding skills in a single coordinate system [8, 9, 11], our geometric feature representation enables encoding motions in multiple coordinate systems, capable of reproducing a wider range of motions. Furthermore, while most prior works either ignore feasibility constraints [12, 8, 9], or enforce feasibility at the expense of inefficiency or suboptimality [13, 14], our structured prior representation gives way to a framework capable of efficiently generating feasibility-constrained optimal robot motions. Lastly, our stable dynamical system representation provides a more expressive representation of reactive motions than existing works [9, 15, 16, 17, 18]. In devising stable dynamical systems, we also go a step beyond traditional approaches by enabling coordinated motions of several robot body parts. Most existing representations of stable dynamical systems [9, 15, 16, 17, 18] are restricted to a single robot body part only.

The aforementioned family of skill representations presented in this work enable sample-efficient learning of diverse skills. Furthermore, the inherent structure in some of the representations is further exploited for computationally efficient skill reproduction via factor

[2]Stability structure guarantees the generated motions to remain bounded. For more details, see Section 7.1.

graphs and computation trees.

1.1 Thesis Statement

The thesis of this work is that **structure in skill representations, dictated by domain knowledge, can enable sample-efficient learning and generalization of diverse robot skills, capable of generating robot motions that are feasible and often coordinated.**

1.2 Contributions

To support our thesis statement, this dissertation makes the following contributions in the field of robot skill learning from human demonstrations:

- **Crowdsourced benchmark for skill learning**: As a prelude to our proposed skill learning approaches, we present an over-arching evaluation of prior skill learning approaches [19]. Instead of evaluating based on hand-specified metrics, which may not fully quantify task execution performance, we carry out a crowdsourced evaluation process based on user ratings. As a result, we identify some key aspects to help guide future research in skill learning. Specifically, in light of statistical analysis, we show that the performance of a skill learning method depends on several factors, including starting positions, task complexity, and demonstrator experience-level. More importantly, since no skill learning technique consistently performed across all the tasks in our evaluations, this benchmarking effort justifies our approach towards developing multiple constrained skill representations, each geared towards a subset of tasks.

- **Multi-coordinate cost balancing for geometry-preserving skills**: We introduce multi-coordinate cost balancing (MCCB) [20] approach, which encodes demonstrations in a cost function. MCCB generalizes a skill by executing motions resulting from solving a constrained convex optimization problem minimizing this cost. Structure is imposed in the representation by realizing the cost as a combination of multiple cost functions. Each

component of the cost function is setup on a particular differential coordinate system, specifying certain aspects of the geometry of desired motions. Further, since the relative importance of each coordinate system in the cost function might be unknown for a given skill, MCCB learns optimal weighting factors that balance the cost function.

- **Probabilistic inference with structured priors for feasible and optimal skills**: This dissertation contributes a probabilistic inference framework called combined learning from demonstration and motion planning (CLAMP) [21]. Under CLAMP, human demonstrations are viewed as governed by a stochastic dynamical system, rolling out which gives a structured prior probability distribution over motions. To generate new motions, the prior is conditioned on the likelihood of starting, ending, or passing through desired states, along with the likelihood of avoiding obstacles in the environment. Additional structure in the prior formulation ensures that the resulting motions also comply with the robot's embodiment constraints. Overall, CLAMP generates motions that are optimal in terms of demonstrations, as well as feasible in terms of environmental and robot's kinematic constraints. The structure in the prior distribution allows representing it on a sparse factor graph, which enables computationally efficient skill reproduction.

  As a follow-up to this work, we also address the problem of learning from demonstrations provided in a cluttered environment [22]. In line of this, we present a weighted learning formulation of CLAMP, whereby the prior is biased towards the demonstrations less likely to be influenced by the presence of obstacles. To further enable improving the prior as more demonstrations are aggregated, an incremental prior learning formulation is also introduced.

- **Diffeomorphically linked stable dynamical systems for reactive skills**: We present an expressive *stable* dynamical system representation grounded in the theory of diffeomorphisms [23]. Unlike their time-dependent counterparts, time-invariant dynamical systems are more suitable for dynamic environments due to their ability to instantly

react to perturbations. To ensure rollouts from the dynamical system remain bounded, we employ a structured representation that enforces stability guarantees. The proposed stable dynamical system is viewed as linked under a smooth, invertible map (i.e. a diffeomorphism) to a simple yet stable latent space dynamical system. The diffeormorphism warps straight-latent motions governed by latent space dynamics, into arbitrarily curved motions, while preserving stability. Thus, we present an expressive formulation for learning diffeomorphisms from data, called *Euclideanizing flows*. Our formulation employs a chain of diffeomorphisms, each parameterized by sparse approximation to kernel machines [24], that can be learned end-to-end. The resulting structured skill representation is called stable dynamical system using Euclideanizing flows (SDSEF).

- **Tree-structured stable dynamical systems for reactive, feasible and coordinated skills**: We introduce tree-structured dynamical system (TSDS) in this thesis, which views a configuration space dynamical system as a weighted combination of multiple *subtask*[3] dynamical systems. Since each subtask dynamical system governs motion of a particular robot body part, our formulation learns simultaneous and coordinated motions of multiple such body parts. Additionally, the structured dynamical system naturally allows enforcing feasibility constraints by embedding robot's kinematic constraints and obstacle avoidance behaviors in the structure. To facilitate efficient dynamical system synthesis, a computational tree structure is employed in this work. We introduce two learning variants for the aforementioned structured dynamical system in this thesis: *(i)* an end-to-end learning approach, which learns all the subtask dynamical systems together, and *(ii)* a independent learning approach, whereby each subtask dynamics is learned independently.

[3]Prior works sometimes use the term *subtasks* to refer to a skill or movement primitive. In this dissertation, a *subtask* represents a subset of motion constraints associated with a single skill.

1.3 Outline of dissertation document

The dissertation is organized as follows. Prior works in the area of skill learning from demonstration are discussed in Chapter 2. Proceeding related works, our contributed family of skill learning methods are divided into three parts. Part I presents our benchmarking efforts evaluating various prior skill learning methods (Chapter 3). Part II studies learning approaches towards time-dependent skill representations and is comprised of Chapters 4, 5, and 6. Multi-coordinated cost balancing (MCCB) is presented in Chapter 4, while combined learning from demonstration and motion planning (CLAMP) is presented in Chapter 5. The weighted and incremental learning formulations of CLAMP are presented in Chapter 6. Part III studies methods for learning time-invariant skill representations and is comprised by Chapters 7, 8, and 9. The Euclideanizing flows formulation for learning stable dynamical system (SDSEF) is introduced in Chapter 7. Chapter 8 presents our tree-structured dynamical system (TSDS) learning formulation with end-to-end learning of constituent subtask dynamical systems. On the other hand, the independent learning formulation for subtask dynamical systems is presented in Chapter 9. Finally, concluding remarks are provided in Chapter 10.

CHAPTER 2
RELATED WORKS

In this chapter, we discuss prior works that address the problem of learning manipulation skills from human demonstrations. Furthermore, we also provide a discussion for placing our contributions in this dissertation in the context of existing works.

2.1 Survey of prior works

As described in Chapter 1, a skill learning technique represents motion constraints associated with a skill in terms of a skill model. The input of a skill model is typically the current state[1], while the output is an action or a sequence of actions (alternatively, next state or a sequence of next states), resulting in a continuous robot motion or trajectory. A significant proportion of existing works also provide current time, in addition to the current state, as input to the skill model, thus making the model time-dependent. Therefore, we categorize prior works in the area into two broad groups: *(i)* time-dependent skill learning methods, and *(ii)* time-invariant (or reactive) skill learning techniques. We discuss existing works from both these aforementioned categories.

2.1.1 Time-dependent skill representations

Perhaps the most commonly employed approaches towards skill learning are based on time-dependent representations. Within this category, we further categorize approaches based on their most distinctive features as: *(i)* time-dependent dynamical system learning methods, *(ii)* cost learning methods, and *(iii)* geometry preservation methods. The aforementioned taxonomy of time-dependent methods however is neither unique nor does it result in mutually

[1] A state includes information about the robot e.g. joint angles, and end-effector position. Additionally it may also include environment information e.g. locations/features of task-relevant objects, and obstacles.

exclusive sub-categories.

Note that certain approaches we consider in this category are not explicitly dependent on time. Instead, these methods are parameterized by a normalized notion of time, given by a surrogate phase variable or arc-length parameter. The implicit time-dependence allows these methods to modulate the speed of reproduced motions if desired.

Time-dependent dynamical system learning methods

The approaches in this category view human demonstrations as rollouts from an underlying time-dependent dynamical system. One of the pioneering works in this group is dynamic movement primitives (DMPs) [25, 11, 10]. At the core of DMPs is a dynamical system made up of a spring-mass damper system, with an additive time-dependent forcing function. The forcing function is defined by a linear function of basis functions, which is learned from human demonstrations. Since its conception, numerous variants of DMPs have been introduced in literature [26, 27, 28, 29], proposing incremental improvements to the initial formulation. A key assumption in DMPs is that a skill can be fully described by a single demonstration. In fact, DMPs can be viewed as a trajectory deformation approach which adapts the demonstration to new starting positions by minimizing a pre-specified Hilbert space norm [30]. Alternative norm specifications have also been presented in literature, capable of deforming a single demonstration in several different ways [31, 32, 33, 34].

Instead of hand-coding a particular generalization criterion, a few extensions of DMPs exploit the statistical properties of *multiple* demonstrations to enable skill-specific generalization via dynamical system representations. Concretely, multiple demonstrations in different situations are collected, and the skill learning technique extracts time-dependent moments (i.e. mean and covariance) of these demonstrations. These moments are typically used in conjunction with DMPs [35, 36, 37, 38, 39] to devise variable stiffness controllers, whereby the stiffness is a function of the mean and covariance. Specifically, a small covariance at a given time instance results in a high instantaneous stiffness.

While the aforementioned methods take advantage of the motion statistics, the motions produced by these techniques are deterministic. Specifically, these approaches assume that in any given context, a single trajectory sufficiently represents the desired behavior. Human demonstrations however are often stochastic in nature, and thus there may exist a distribution over trajectories that execute a skill in a particular context. Learning trajectory distributions provide additional flexibility in adapting the skill to new contexts since there are multiple options to choose from. A few approaches in this regard learn a stochastic dynamical system or stochastic control law, which results in a distribution over possible trajectories. While Gaussian processes (GP) have been employed to parameterize the aforementioned stochastic control law [40], a more parsimonious representation was introduced by probabilistic movement primitives (ProMP) [12, 41]. ProMPs represent each demonstration in weight-space, and estimates a Gaussian distribution over these weights. A stochastic control law is then derived utilizing this weight-space distribution. ProMPs have been shown to be capable of adapting the skill to new via-points or goals by way of conditioning the weight distribution. Based on a similar premise as ProMPs, other methods can also be found in literature that learn to reproduce trajectory distributions, including kernelized movement primitives (KMP) [42]. Unlike ProMPs, but similar to GPs, KMPs employ a kernelized treatment of the trajectory distribution learning problem. Thus, while KMPs do not rely on explicit selection of basis functions, it suffers from the problem of high computational complexity associated with kernel methods [24].

Cost learning methods

Methods belonging to this category assume that the human demonstrations are optimizing a latent cost (or reward) function. Therefore, these methods aim to extract this cost function from human demonstrations. We highlight that some approaches in this group can also be classified as inverse optimal control (IOC) methods [43]. However, the category of IOC methods has a wider scope than this work. Specifically, in IOC methods, the learner does

not have to necessarily be a robot manipulator, while the demonstrator does not have to be limited to a human expert either. This dissertation is instead focused on approaches that are geared towards learning robot manipulation skills from human experts.

Among the most widely used approaches in this regard are methods that learn a cost function based on statistics of motions. As discussed earlier, motion statistics often refer to the mean and covariance of the demonstrations. One of the earliest approaches grounded on the statistical properties of motions was proposed in [8]. This approach finds a time-conditioned Gaussian state distribution by employing Gaussian mixture models (GMM) followed by Gaussian mixture regression (GMR). The conditional distribution gives way to a cost function which penalizes deviations from the time-varying mean weighed by the inverse of the time-varying covariance. The premise behind this cost function is that a low covariance at a given time instance corresponds to a strong desire for staying near the mean of demonstrations at that instance. An extension to this approach was later proposed, called task-parameterized GMMs (TpGMM) [44]. Instead of a single conditional distribution, TpGMMs find conditional distributions in multiple reference frames, each associated with a target object. Thus the cost given by TpGMMs is a function of its environment, enabling better generalization. TpGMMs result in a quadratic cost function, which can be used in an LQR setting to derive a feedback control law or dynamical system for skill reproduction [45]. However, in the presence of other non-convex costs or non-linear constraints, trajectory optimization routines discussed below are more suitable.

Conventional trajectory optimization methods aim to find robot motions satisfying hand-coded criteria e.g. smoothness, obstacle avoidance etc. Skill learning methods based on trajectory optimization, instead seek to learn the criteria from demonstrations. Specifically, trajectory optimization based learning formulations learn a potentially non-quadratic cost function, which is then employed in a constrained optimization setup. The learnable cost function is typically a function of several non-linear features of the environment and/or robot state [46, 30, 47, 48, 49]. Instead of providing a control law/policy, trajectory optimization

methods output an entire trajectory that can be passed to a low-level tracking controller on the robot. Since the trajectory is computed beforehand, the motions can not react to perturbations. However, trajectory optimization allows employing additional non-quadratic costs and highly non-linear constraints (e.g. obstacle avoidance), making them applicable to a broader set of problems.

Geometry preservation techniques

Another stream of approaches found in literature, represent skills in terms of geometric features of motions. The premise here is that the shape of motions is most critical in executing a given skill. A few approaches in this sub-group employ Frenet-Serret formulas to represent geometric properties of motions [32, 33, 50]. Frenet-Serret formulas describe the motion of a particle traversing a curve in 3D space, parametrized by arc-length [51]. Notable in this category is an approach called trajectory learning using generalized cylinders (TLGC) [50]. TLGC finds a generalized cylinder, or simply a tube in space, which tightly fits multiple demonstrations. To reproduce the skill, TLGCs utilize a ratio rule which forces the motions to follow the shape of the cylinder's closest boundary.

2.1.2 Time-invariant skill representations

While time-dependent representations of skills have shown promising results on a variety of tasks, they also carry certain disadvantages. First, time-dependent approaches can not always instantly react to positional perturbations, making them inapplicable to dynamic environments. Some time-dependent methods that derive a feedback control law or dynamical system can indeed be classified as reactive (e.g. DMPs and ProMPs). However, these approaches are still susceptible to suffer in the presence of temporal perturbations, brought about by situations where the reproduction time horizon may vastly differ from that during demonstrations. On the other hand, time-invariant representations are known to relatively robust against spatial and temporal perturbation [9]. Unlike time-dependent methods, the

category of time-invariant skill representations is dominated by dynamical system based methods.

Stable time-invariant dynamical system learning methods

Time-invariant dynamical systems also have their fair share of challenges. Since time-invariant dynamical systems can be rolled out for long time horizons, the resulting motions can diverge and blow up to infinity. To mitigate this challenge, it is common to enforce global stability properties in the the dynamical system parameterization. The resulting dynamical system, also referred to as a *stable dynamical system*, is guaranteed to result in motions that always remain bounded.

Over the past two decades, a number of approaches have been proposed towards learning stable time-invariant dynamical systems. In literature, first-order dynamical systems (mapping positions to velocities) are mostly employed, however second-order representations (mapping position-velocity pairs to accelerations) are also applicable. One of the earlier approaches in this regard is SEDS [9]. SEDS realizes dynamics as a mixture of linear dynamical systems, learned using GMMs. The global stability constraints in SEDS simplifies to enforcing stability of each linear system. This simplification however comes at the expense of accuracy. Specifically, SEDS assumes the demonstrations comply with a quadratic Lyapunov (or Energy) function[2], and thus is restricted to motions which monotonically converge to the goal over time. A relaxation to the stability criterion was proposed in CLF-DM [15], which instead assumes a Lyapunov function in the form of a weighted sum of asymmetric quadratic functions (WSAQF). The parameters of the Lyapunov function are also learned, although independently from th dynamical system, and then used in an online fashion to generate stabilizing controls. The online correction scheme may interfere significantly with the learned dynamics [18] and thus cause inaccuracies. A more recent approach CDSP [16] instead enforces incremental stability, a notion concerned with relative

[2]Lyapunov functions are commonly employed in controls literature for specifying a notion of stability [52]

displacement of motions. Instead of learning a Lyapunov function, CDSP proposed learning a positive-definite contraction metric. Positive-definiteness however still has to be ensured globally, to realize which CDSP restricts the class of contraction metrics to (potentially limiting) sum of squared polynomials. Furthermore, in a similar fashion to SEDS, CDSP relies on a potentially restrictive dynamical system formulation made up of a mixture of linear dynamical systems.

There are a few approaches that instead learn a stable dynamical system using diffeomorphisms, which are smooth invertible maps. These approaches are based on the premise that a non-trivial dynamical system is diffeomorphically related to a simpler dynamical system in a latent space. Therefore, these methods learn diffeomorphisms from human demonstrations. However, learning diffeomorphisms also requires enforcing the function to be invertible, making the learning problem non-trivial. One approach [17] in this regard, learns a diffeomorphism composed of *locally weighted translations*, while hand-specifying the latent space dynamics. The authors of this approach however only showed it to work with a single (or an average) demonstration. Another approach, namely τ-SEDS [18], learns diffeomorphisms from multiple demonstrations. In τ-SEDS, a diffeomorphism is given by the square-root of a WSAQF. However, to account for the limited flexibility of WSAQF, the latent space dynamics are also learned, instead of hand-specifying, using SEDS.

2.2 Our contributions in the context of prior works

2.2.1 Multi-coordinate cost balancing

Each of the representations mentioned in the previous section, encode the demonstrations in a predefined space or coordinate system. In other words, a single best coordinate system for any given skill is assumed to both exist and be known. As shown in Chapter 4, the assumption of existence of a single best coordinate system does not hold for all skills. To contend with this challenge, our multi-coordinated cost balancing (MCCB) method (Chapter 4), represents motions in multiple differential coordinates. MCCB is also a cost learning approach which

exploits the statistics of motions. However, unlike most existing statistical cost learning methods [44], the cost function in MCCB is not limited to statistics of motions in Cartesian coordinates only. Instead, MCCB sets up the cost as weighted combinations of costs in differential coordinate systems. Each differential coordinate represents certain geometric aspects of the motions, drawing connections to geometry preserving approaches mentioned earlier. Most geometric representations [50, 32, 33], however focus on a subset of these geometric aspects, dictated by the Frenet-Serret formulas. Furthermore, instead of hand-tuning the relative of influence of each coordinate system in the cost function, MCCB also learns to balance these influences from demonstrations. To generate new motions, MCCB employs a trajectory optimization formulation. The framework in [46], complementary to our approach, also utilizes a blended cost function, the construction of which is guided by probability distributions learned from the demonstrations. This framework incentivizes factors such as smoothness, manipulability, and obstacle avoidance, but is also restricted to the Cartesian coordinate system.

2.2.2 Probabilistic inference with structured priors

Most existing skill learning techniques [10, 8, 9] are limited to learning skill models that represent motion constraints in either the configuration space of the robot, or the task space (typically dictating the robot's end-effector motions). While configuration space skill models can be beneficial in settings where there is a preference on certain robot configurations, a large variety of robot skills are naturally expressed in the task space [53]. To execute a skill, the task space motions given by the skill model are mapped to corresponding configuration space motions by employing either an inverse kinematics solver, a motion planner or a trajectory tracking control law. However, such a post-hoc approach may not result in desirable robot behaviors. Specifically, the motions generated by the skill model may not be reachable by the robot due to its kinematic constraints, or presence of obstacles in the path. Even if a post-hoc routine finds feasible configuration space motions, the resulting motions may

not satisfy the task space constraints associated with the skill anymore, thus introducing suboptimality. To cater to the aforementioned challenges, our probabilistic inference formulation, CLAMP (Chapter 5), introduces a unified formulation that provides configuration space motions while simultaneously accounting for the feasibility and optimality criteria. Similar to ProMPs, CLAMP also assumes that the demonstrated trajectories are governed by a time-dependent stochastic dynamical, which results in a trajectory distribution. However, unlike ProMPs, CLAMP carries out probabilistic inference directly in the space of trajectories, instead of weight-space. The original ProMPs formulation [12] neither accounts for obstacle avoidance, nor does it map task space skills to configuration space. A proceeding formulation adds obstacle avoidance capabilities to ProMPs [13], but rather in (partially) redundant two-step process. Specifically, the trajectory distribution is first adapted offline for obstacles, and then re-adapted online for new start/goal states. Moreover, this formulation has not been shown to work for multi-DOF manipulators which generally require efficient collision checking.

CLAMP exploits duality between probabilistic inference and trajectory optimization to carry out efficient skill reproduction, drawing connections to motion planning, specifically GPMP2 [54]. There also exist other works that make similar connections. One of the earlier approaches that saw this connection was presented in [14]. This approach employs a sampling-based motion planner to adapt the output of a learned statistical model to avoid obstacles. As discussed before, such a hierarchical approach induces redundancies in the motion generation routine by assuming the two constituent steps as independent. A couple of other similar approaches [30, 46] extend the CHOMP motion planner for skill learning. CLMAP inherits the computational advantages of GPMP2 and thus provides orders of magnitude faster convergence than CHOMP [54]. Furthermore, unlike CHOMP-based methods, CLAMP also allows extracting motion constraints on higher-order time derivatives of motions (i.e. velocities and accelerations).

2.2.3 Diffeomorphically linked stable dynamical systems

Our stable dynamical system learning formulation in Chapter 7 learns diffeomorphims from demonstrations. Specifically, we view the stable dynamical system as linked under a learnable diffeomorphism to another simple hand-specified dynamical system. Unlike traditional stable dynamical system formulations [9, 15, 16], both the dynamics and the Lyapunov function in our approach are bi-products of the diffeomorphism itself, thus removing the need for directly learning a dynamical system and/or stability criterion. In contrast to prior works on learning diffeomorphisms [17, 18], we paramterize diffeomorphisms by using more expressive function approximators including kernel methods [55] and neural networks [56].

Our diffeomorphism learning approach builds on normalizing flows [57, 58], which have recently been successfully used for density estimation [59]. The goal of normalizing flows is to map a simple base distribution into a complicated probability distribution over observed data by applying the change of variable theorem sequentially. Our problem is similar to the normalizing flows problem as we seek to map simple straight-line motions to complicated motions captured from human demonstrations. However, our problem is fundamentally different in two ways. First, normalizing flows only require the mapping to be bijective, a property less strict than diffeomorphism. Second, normalizing flows are concerned primarily with mapping scalar functions to scalar functions (i.e. probability densities). In our method, we seek to map vector fields to vector fields.

2.2.4 Tree-structured stable dynamical systems

As previously discussed, while robot motion generation is concerned with finding a series of configuration space commands that execute a task, the motion constraints are often naturally expressed in a different space, called a task space. Existing approaches towards learning time-invariant stable dynamical systems [9, 15, 60, 17, 16] are generally limited to learning in task spaces that dictate motions of a single body part of the robot only

(e.g. the end-effector). We argue that certain tasks, and hence the constituent skills, may enforce multiple motion constraints potentially on several robot body parts. Thus a task can be viewed as made up of multiple subtasks, whereby each subtask is assigned to a particular robot body part. Simultaneous and coordinated execution of all the subtasks is required for successful task execution. To address this challenge, we employ a structured dynamical system representation similar to [61]. This structured dynamical system is defined in the configuration space and is a weighted combination of several subtask dynamical systems setup on a computation tree. We learn the constituting subtask dynamical systems from human demonstrations. In Chapter 8 and Chapter 9 we present two such learning formulations. Since the subtask dynamical systems are linked to the configuration space by the robot's forward kinematics map, the overall structured dynamical system naturally encodes robot's kinematic constraints. Furthermore, some of the subtask dynamical systems can also be hand-specified to enforce safety guarantees (e.g. obstacle avoidance). Lastly, our choice of subtask dynamical system parameterization always guarantees the resulting structured dynamical system to be stable. There are prior works that have explored learning a combination of multiple dynamical systems in either robot's configuration space only or in conjunction with its workspace [62, 63, 64]. These methods however employ time-dependent dynamical systems, thus suffering from the earlier mentioned generalization challenges associated with time-dependent models. Furthermore, having time as an input to the skill model simplifies the learning problem to certain extent, since time acts as a coupling term to enforce coordination between multiple time-dependent dynamical systems.

Part I

Benchmarking Skill Learning from Demonstration

CHAPTER 3

BENCHMARK FOR SKILL LEARNING FROM DEMONSTRATION: IMPACT OF USER EXPERIENCE, TASK COMPLEXITY, AND START CONFIGURATION ON PERFORMANCE

As discussed in Chapter 2, there exist a variety of approaches aimed at learning skills from human demonstrations. While the relative advantages and disadvantages of the commonly-used approaches might be known within the LfD community, there do not exist comprehensive guidelines for non-experts outside the community to assist in using these methods. Specifically, it is critical that empirical studies be performed to compare the relative strengths and identify the short-coming of existing techniques.

From the perspective of an end user, there are multiple desirable properties that a skill learning approach should have, including the ability to:

1. learn skills from demonstrations provided by end users irrespective of all experience levels, with minimal information overload on the user,

2. learn a variety of skills, which may differ in the level of complexity, and

3. reproduce a learned skill in scenarios similar to or different from those encountered during demonstrations.

Instead of evaluating on the basis of aforementioned criteria, existing techniques are often tested in isolation by experts for only a specific set of tasks using a set of hand-picked metrics. Comprehensive surveys on skill learning from demonstrations [65, 1, 66, 67, 2] do exist, but they mainly focus on summarizing existing approaches, proposing taxonomy, and reporting challenges associated with employing skill learning from demonstration approaches in practice. There is a need to supplement these surveys by comparing and evaluating skill learning approaches across several variables that exist in the real world.

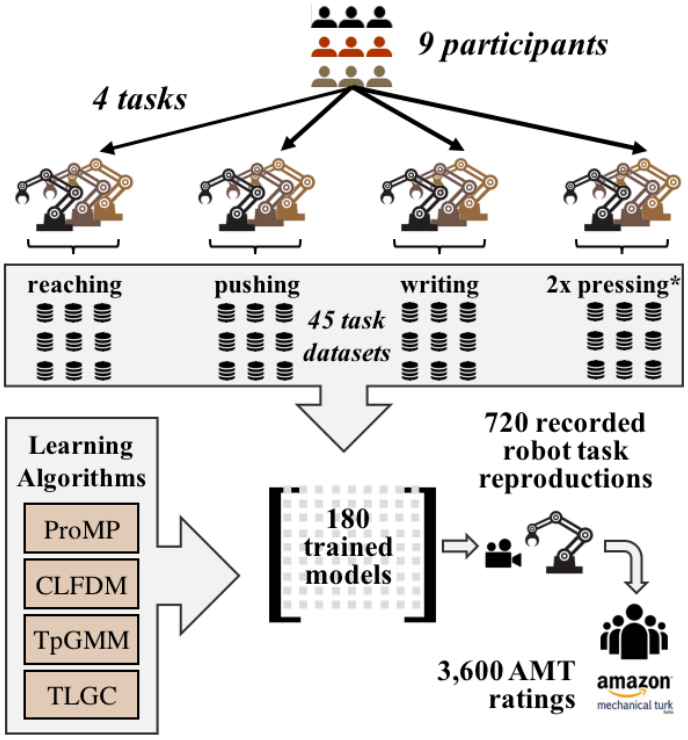


Figure 3.1: Overview of the benchmarking study experimental design.

Therefore in this work, we evaluate the performance of multiple skill learning approaches and examine how the i) complexity of the task, ii) expertise level of the human demonstrator, and iii) starting configuration of the robot affect performance of each technique. To perform this evaluation, we collected data from nine participants across four different manipulation tasks with varying starting conditions. The resulting demonstrations were used to train 180 task models. Each of the resulting models was then executed on a Rethink Sawyer robot, resulting in 720 videos of robot task reproductions. Finally, we obtained 3600 Amazon Mechanical Turk ratings to evaluate the robot's performance in the videos. Figure 3 provides an overview of our experimental procedure. Additionally, we present an evaluation based on quantitative error metrics obtained by assessing the similarity between the reproduced trajectories and the demonstrations.

Our results show that the performance of the skill learning approaches — irrespective of their underlying representation — is generally predictable when the new starting condition

is closer to the starting position of demonstrations. However, as the generalization scenario differs from the demonstrations, the consistency of an approach's performance across generalization scenarios is highly dependent on the task constraints. Furthermore, we also find that the performance of a given skill learning method is correlated with the experience level of the human providing demonstrations. Lastly, we found that commonly used performance evaluation metrics such as mean squared error are not always able to correctly predict the generalization performance of an approach.

Prior work by Lemme *et al.* [68] contributed a valuable benchmarking framework to evaluate the performance of reaching motion generation approaches on a 2D handwriting dataset. Their study evaluates the algorithms' generalization ability in simulation and presents performance metrics on a small scale. Our study is more comprehensive: it covers multiple tasks, incorporates diverse constraints and variables, and is performed on a physical robot. To our knowledge, no benchmarking study exists that independently evaluates a wide range of task execution conditions. Additionally, no prior studies report human ratings of task performance.

We intend for this work to be used by those who study LfD by acting as a reference for experimental design, evaluation metrics, and general best practices. The full dataset of demonstrations, videos of executions, and accompanying evaluations have been made publicly available to aid future benchmarking efforts[1].

3.1 Experimental Design

This section provides an overview of our experimental design process, including the selection of algorithms, the choice of tasks, human participant selection, as well as methodology for data recording and model evaluation.

[1]Project website: https://sites.google.com/view/rail-lfd. Accompanying video with highlights of experimental process and results: https://youtu.be/KuUYxogR4WU.

3.1.1 Techniques Selected for Comparison

In Chapter 2, we discussed four categories of skill learning approaches, which included: *(i)* time-invariant stable dynamical systems [9, 15, 16], *(ii)* time-dependent dynamical systems [11, 12], *(iii)* cost learning methods [8, 44], and *(iv)* geometry preserving methods [32, 31, 69]. The approaches evaluated in this work were chosen to represent each of the aforementioned groups, with one approach per group. Below, we provide a brief description of each method; see references for full details.

CLFDM [15] – A time-invariant approach which learns a dynamical system mapping positions to velocities. It is assumed here that the final positions of the demonstrated motions are centered at a single goal location, and the dynamical system rollouts are guaranteed to coverge at the goal.

ProMP [12] – A time-dependent approach which describes demonstrations as a set of weights. ProMP derives a stochastic control law as a function of these weights, rolling out which reproduces a distribution over trajectories.

TpGMM [44] – An approach which encodes the statistical features of demonstrations as a joint probability density over time and states. The version of TpGMM employed in this work finds new trajectories by solving an LQR problem, resulting in a control law which tracks the demonstration mean with a time-varying gain.

TLGC [50] – An approach which encodes the geometric features of the demonstrations. TLGC bounds demonstrations by an arc-length parmeterized *generalized cylinder*. Given a new initial position, a new trajectory is found by tracking the curvature of the nearest boundary of the cylinder.

While many other skill learning approaches exist, we have selected these four approaches because they are well-known, commonly used, or are most mature based on incremental improvement on prior work.

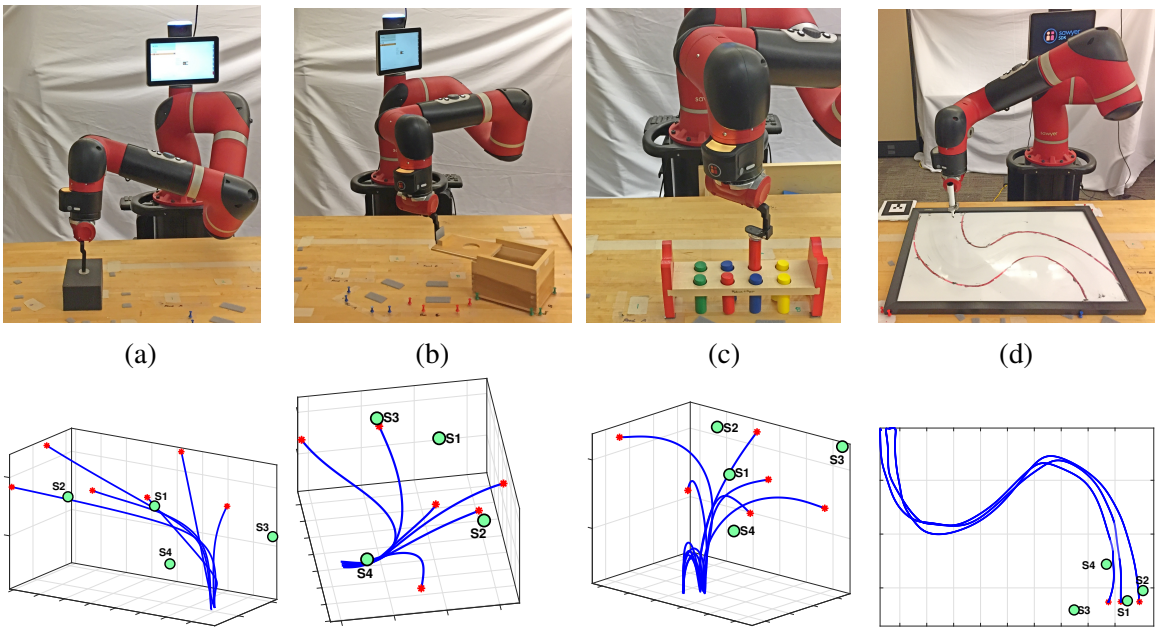


Figure 3.2: From left to right, visualizations of the *reaching*, *pushing*, *pressing*, and *writing* task. The bottom row plots example demonstrations (blue) for each tasks. The red dots denote demonstrated starting positions while the green circles represent *new* initial positions selected for evaluating skill generalization.

3.1.2 Robot Tasks

We selected four tasks (Fig. 3.2) each of which contains unique properties representing different level of motion constraint complexity. Human demonstrator ability was kept in mind such that users could execute the tasks with minimal experience on the robot. To minimize any novelty effect that may exist, the demonstrators performed an additional practice task on the robot before moving on to the actual tasks. Following are brief descriptions of the tasks.

- *Reaching* (Fig. 3.2a) – Move toward and touch the circle on the gray block . This task poses a hard constraint on the end position.
- *Pushing* (Fig. 3.2b) – Push the box lid closed. Comparing to the previous task, this task is constrained in the direction of motion towards the end. The position constraint for the endpoint is not as hard as in the *reaching* task.
- *Pressing* (Fig. 3.2c) – Push down peg $\#1$ and then peg $\#2$. Compared to *pushing*, this task is more constrained in both the direction of motion as well as end-positions.

- *Writing* (Fig. 3.2d) – Draw an S-shaped curve on the whiteboard . Compared to other tasks, this task requires a harder constraint on the direction of motion to follow the curvature of the shape.

3.1.3 Participant Selection

Skill reproduction can be significantly affected by the performance of the demonstrator [1]. Knowing the extent of which performance is affected by demonstrator experience level could be a useful metric in designing future experiments. Therefore, we chose to include demonstrators with different experience levels in our experiments.

We recruited nine participants with different levels of robotics experience from the Computer Science and Engineering community at Georgia Tech. Three participants with *Low* experience had no prior interaction with a robot. Three participants with *Medium* experience had worked with robots but had no experience in robot manipulation, and particularly no experience in kinesthetic teaching. Three participants with *High* experience had some prior experience with LfD and kinesthetic teaching.

3.1.4 Data Recording

Data collection with participants followed an IRB-approved human subjects study protocol and participants were compensated with a $10 gift-card. Upon arrival, participants were briefed about the goals of the study and taught to interact with the robot using a practice task (i.e., pushing a toy car across the table using the robot's end-effector).

Participants received written instructions that included a verbal description and photos of the goals of each task[2]. This ensured the consistency of the guidelines across all participants and evaluators. Before recording, the robot and the target object was initialized to one of the pre-defined starting configurations. The robot was then put in gravity-compensation mode, and the participant kinesthetically guided the robot to accomplish the task. Finally,

[2]Example: Fig. 3.2(a) accompanied by the instruction, "The robot finger-tip should touch the small circle on the gray block".

the recording was stopped at the participant's command. Each recorded demonstration consisted of the robot's end-effector trajectory alongside the location of target object in the environment.

In order to assess the quality of the demonstrations, we provided the participants with a visualization of the recorded trajectory in ROS RViz. Participants were allowed to perform multiple executions of the task until they were satisfied with the quality of the data; we kept only the final execution. In total, the participants provided three demonstrations for the *writing* task with three different starting positions. For the remaining tasks, six demonstrations (3 starting positions $\times$ 2 object locations) per participant were collected. This resulted a total of 21 demonstrations per participant. Fig. 3.2 (*bottom*) shows an example set of demonstrations, transformed such that the origin is at the target object location.

3.1.5 Model Evaluation

From the collected demonstrations, we constructed 45 task datasets. Each dataset included all demonstrations of a specific task (four tasks) performed by a specific participant (nine participants). Note that each participant was asked to demonstrate the *pressing* task twice each time under a different condition (see Section 3.2.2 for more detail), resulting in 9 participants $\times$ 5 tasks $=$ 45 datasets.

Each of our four algorithms was then trained on each of the 45 datasets, resulting in 180 task models (one per participant-task-algorithm combination). For evaluation, we executed each of the 180 models under four different starting conditions on a Sawyer robot, resulting in 720 video recordings of robot task executions. To obtain a final evaluation of the robot's performance in each of the videos, we employed five AMT [70] workers to evaluate the quality of each video, resulting in approximately 3600 performance ratings.

3.1.6 Amazon Mechanical Turk Evaluation

Evaluations were crowdsourced using the Amazon Mechanical Turk (AMT) platform in order to obtain results quickly and without bias. To ensure that AMT workers evaluating the robot had a consistent understanding of the task goals, workers were shown the same set of instructions as those given to the study participants (i.e., task demonstrators). For each video of the robot's task execution, AMT workers were asked to answer the following questions:

Q1. Please rate the extent to which you agree with the statement: *"The robot efficiently and safely completed the goal(s) of the task."* (Strongly agree; Agree; Disagree; Strongly disagree).

Q2. Please also specify which of the following contributed to your rating in the previous question. (Check all that apply)

- The robot failed to achieve the goals of the task (*incomplete*).
- The robot performed unnecessary motion (*inefficient*).
- The robot acted in an unsafe manner (*unsafe*).

Each video was evaluated by five AMT workers and an overall *rating* per video/execution was calculated by taking the median of the responses to the first question (Q1). To get a quantitative measure of the evaluator rating, we mapped the answers to numerical values: Strongly agree $= 3$, Agree $= 2$, Disagree $= 1$, and Strongly disagree $= 0$. We consider a task reproduction to be acceptable to the evaluators if the rating is 2 or above. Answers to the second question were only considered if the participant selected a rating below "Strongly agree" in response to the first question.

The selected keywords, *incomplete*, *inefficient*, and *unsafe* in Q2, are suitable to define the characteristics of the task execution quality from an end user's point of view. Our reasoning is that a robot that cannot complete a task efficiently can impose great burden on the user, and a successful human-robot team requires a smooth and predictable task execution.

3.2 Data Processing and Validation Scenarios

This section provides an overview of the data processing and parameter tuning methods used in our evaluation, as well as the design of the starting robot configurations used in evaluating the generalization of the chosen approaches.

3.2.1 Data Preprocessing

The recorded robot end-effector motions were often found to be affected by high-frequency noise, caused either by sensor inaccuracies or by the demonstrator's inability to guide the robot through smooth paths. To address this, we applied a low-pass moving average filter on the raw data. Additionally, we estimated the velocities of the end-effector using 1st-order finite differencing. Finally, for methods that require time-aligned trajectories, we also warped the speed of demonstrations such that they end at the same time. We employed dynamic time warping (DTW) [71] routine for this purpose.

3.2.2 Motion Segmentation for Pressing Task

Unlike the other tasks, *pressing* can be seen as two separate tasks or primitives; that is, pressing the first peg followed by pressing the second peg. We assume that considering these two segments as one was likely to adversely affect the performance of some of the approaches. Hence, to ensure fairness in our comparisons, we conducted experiments of the *pressing* task once *without* segmentation and once *with* segmentation. We performed an additional pre-processing step of motion segmentation [5, 72, 73, 74] for the *pressing* task and made a separate dataset for this variation. Specifically, we passed the demonstrated trajectories through a changepoint detection routine [75], which segments the trajectories where peaks are encountered in the normalized velocities. The output was further manually checked to ensure good segmentation. For a given approach, we trained a model per segment, reproduced the task segments separately, and stitched the reproduced segments together to

be executed by the robot as one trajectory. Throughout the chapter we clarify which variant of *pressing* is being used, and we evaluate the effect of segmentation on performance in Section 3.3.3.

3.2.3 Parameter Tuning

Our work is motivated by potential real-world applications of skill-based LfD methods, such as factory operation. To mimic a realistic operational context for the robot, we chose to use only a single common set of parameters for each algorithm. More specifically, we tuned a parameter set for each *algorithm*, but did not tune unique parameters *per task*, since this would be impractical in real-world settings. We performed the tuning process on the LASA dataset [76] and a small randomly selected subset of robot demonstrations. We manually tuned the parameters of each method until we observed consistently good performance across the test set.

3.2.4 Generalization Scenarios

To validate the generalizability of the learned models, each task model was evaluated on four new initial end-effector configurations relative to the target object, $S1$-$S4$. Figure 3.2 (bottom row) visualizes the initial positions for each task. $S1$ was selected to be within 90% confidence interval around the mean of the initial positions of the demonstrations. $S2$-$S4$ were selected outside this range, such that $d(S3) > d(S2) > d(S1) > d(S4)$, where $d(\cdot)$ denotes the Euclidean distance to the target object. $S2$ and $S3$ were chosen to be farther away from the target object, while $S4$ was chosen to be closer to the object.

Note that since each technique was trained on and hence generated robot's end-effector motions, an inverse kinematics solver was employed post-hoc to generate corresponding joint-space commands.

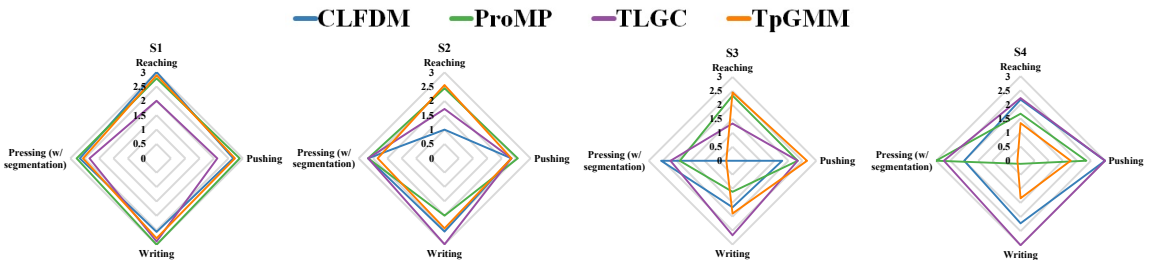


Figure 3.3: Radar plots reporting average ratings against executed tasks, for a particular starting position. The average is computed over nine datapoints corresponding to the nine recorded videos, where each video represents a model query at the given generalization scenario.

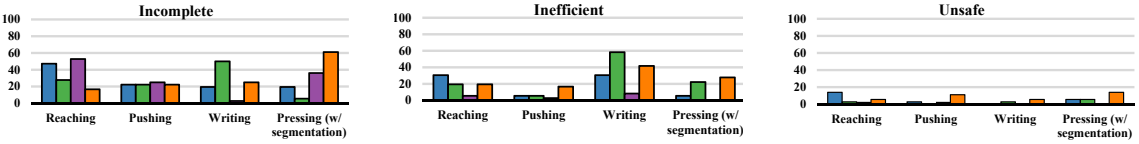
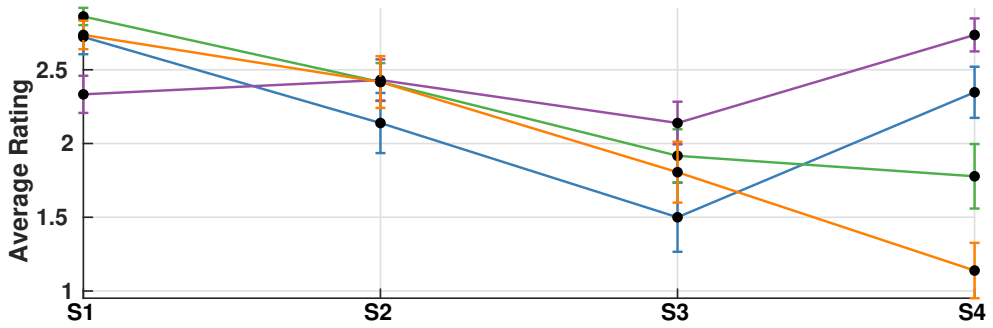


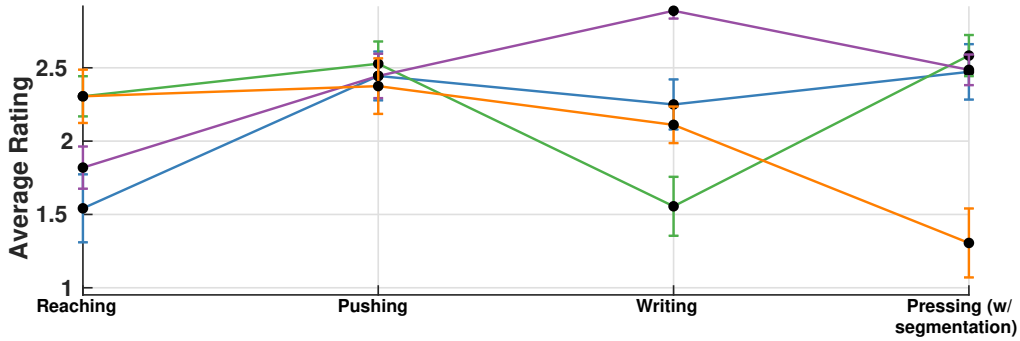
Figure 3.4: Bar charts reporting subjective user feedback, whereby each bar represents the number of times a given reason was cited – as a percentage of the total number of robot executions for a given task-algorithm combination.

3.3 Generalization Performance across starting positions and tasks

In this section, we study how the average rating for each skill learning method varies across two independent variables: (1) starting position and (2) task. The results are visualized as radar plots in Figs. 3.3a through 3.3d, where each radar plot reports the average ratings against the executed tasks, for a particular starting position (e.g., S1). The average ratings are computed over nine datapoints corresponding to the nine recorded videos, where each video represents a model query at the given generalization scenario. Also reported in Fig. 3.5(*top*) are ratings, further averaged against all tasks, per starting position, while Fig. 3.5(*bottom*) plots ratings against tasks, averaged against all starting positions. Furthermore, we also provide an analysis of the feedback provided by the evaluators as answers to Q2 in Section 3.1.6. Fig. 3.4 reports the feedback – where the bar charts represent the number of times a particular reason was cited for a given generalization scenario – as a percentage of the total number of robot executions/videos for that same scenario.



(a) Average user rating against starting positions.



(b) Average user rating against task.

Figure 3.5: Trends of average user rating against two independent variables.

3.3.1 Trends across starting positions

We see larger variations in average performance of approaches across tasks when the distance between the robot's starting position and the target location is progressively increased (S1 through S3), as shown in Fig. 3.3a through 3.3d. In general, as evident from Fig. 3.5a, we noticed worsening performance with increasing distance to the target. This is particularly noticeable for the *writing* and *reaching* tasks in Figs. 3.3b and 3.3c. However, when the target distance is significantly decreased (S4), CLFDM and TLGC performed consistently in an acceptable manner across tasks, while ProMP and TpGMM generally under-performed. Overall, TLGC was least affected by the changes in starting positions for the *pushing*, *writing*, and *pressing* tasks. However, on the *reaching* task, where the other approaches performed generally well, TLGC performed the worst and often at an unacceptable level.

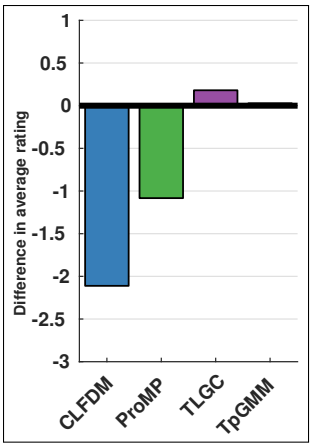
3.3.2 Task-wise evaluation and subjective user feedback

Analysis in this subsection is based on Fig. 3.5b in conjunction with subjective user feedback from Fig. 3.4. The video accompanying this work shows some of the failure/success cases mentioned here.

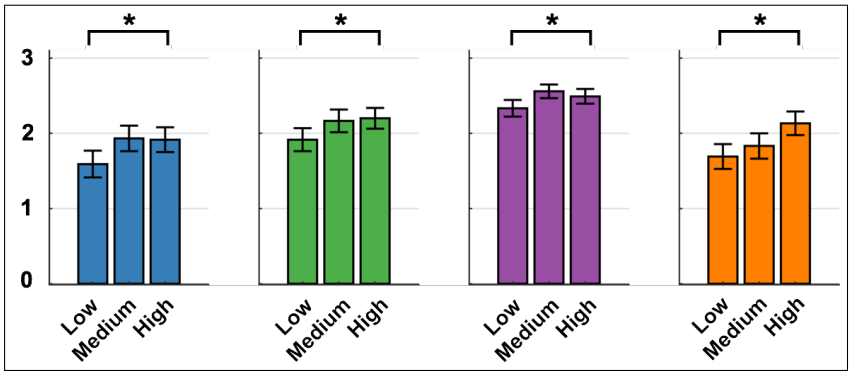
For the *reaching* task, TLGC is hypothesized to have accrued low ratings due to robot executions which often stopped a short distance from the target. Users often marked these executions as *incomplete*. CLFDM was found to not generalize well for starting positions S2 and S3 which are farther from the target, and had a high percentage *incomplete*, *inefficient*, and sometimes *unsafe* ratings. We hypothesize that this is due to often long, unpredictable paths generated by CLFDM. Furthermore, due to this unpredictability, the robot often collided with the table and hence failed to complete the task; thus the evaluators frequently marked the executions as *incomplete* and *unsafe*.

On *pushing*, although all approaches were consistent on average across starting positions, we noticed several failure cases. TpGMM was sometimes perceived as *inefficient* and *unsafe* when starting too far from (S3) or too close to the box (S4). During some of these executions, the robot pushed farther than necessary into the box and dismounted it.

For *writing*, only TLGC generalized across starting positions. CLFDM was observed to be the second most consistent across starting positions, except when starting away from the final position (S3). CLFDM often drew a longer L-shaped curve instead of the desired S-shape, which was marked as *inefficient* and *incomplete* although it was mostly smooth and safe. Executions by ProMP were frequently marked as *incomplete* and *inefficient* since it was often observed to draw non-smooth curves when starting farther away, i.e. S2-S3 or illegible shapes when starting closer (S4). When starting from S3, TpGMM was also found to draw an S-shaped curve with relatively sharp edges. Lastly, for both TpGMM and ProMP, the robot was frequently observed to go back a short distance from S4 before drawing, often penalized by evaluators for being *inefficient*. There were also a few occasions, where TLGC produced some extraneous motion in the beginning, however this was often not penalized



(a)



(b)

Figure 3.6: *(a)*: Difference in ratings on the *pressing* task with and without motion segmentation pre-routine. *(b)*: Average ratings grouped by algorithm (CLFDM, ProMP, TLGC, TpGMM) against the experience level of the demonstrators.

by evaluators.

For the *pressing* task, TpGMM was severely affected by variations in starting positions. TpGMM frequently carried out extraneous motions for S3 and S4, often failing to press any of the pegs. Moreover, TpGMM occasionally followed a pressing motion but stayed above the pegs. Such executions were often rated *incomplete* and *inefficient*. ProMP was sometimes marked *inefficient*, which can be attributed to jerky, extraneous motions when started far from the pegs.

3.3.3 Effect of motion segmentation

We conducted an additional evaluation to test our hypothesis regarding the adverse effect of learning on unsegmented data on the *pressing* task's performance. We trained each algorithm on unsegmented data and performed the same crowdsourced rating in Section 3.1.6. Fig. 3.6a shows a bar chart comparing performance with vs. without segmentation. Each bar shows the average rating *without* motion segmentation subtracted from the average rating *with* the segmentation routine. We observed that ProMP, and especially CLFDM, suffer significantly when segmentation is not used. This is an expected result for CLFDM, which

is incapable of learning self-intersecting motions [15]. In fact, this property is common across all LfD approaches that learn firs-order time-invariant dynamical systems [77, 18, 17, 9].

3.4 Performance Across Experience Level

In this section, we present an analysis on the dependence of the evaluator ratings, averaged over all the tasks and starting positions, on the experience level of the demonstrators. Fig. 3.6b provides a visualization of the results.

All the methods show similar increase in average rating from low to high experience when each algorithm is individually observed across experience levels. To corroborate this trend, we also carried out a two-way ANOVA analysis for the approaches against the experience levels. We found that the experience level has a statistically significant effect on average ratings ($p = 0.0389 < 0.05$), while no statistically significant interaction effect was found between the two variables ($p = 0.95 > 0.05$). We further carried out Tukey's range test, which determined that there was a statistically significant effect on performance between the low and high experience levels ($p < 0.05$). However, no statistically significant difference in performance was found for low and medium, or medium and high experience levels. A secondary analysis was also carried out on the reasons the evaluators provided for their ratings. This showed that there was a statistically significant difference between user experience levels low and high ($p < 0.05$) for a video being marked as inefficient. This means that the evaluators considered the lower-rated videos corresponding to the low experience demonstrators to be more inefficient on average.

In conclusion, we see that higher demonstrator experience positively affects performance across all algorithmic conditions. Interestingly, little difference in performance is observed between participants with high and medium levels of experience (participants with kinesthetic teaching experience vs. participant with general robotics experience). This observation indicates that having prior knowledge about robots, sensing, or sensitivity to

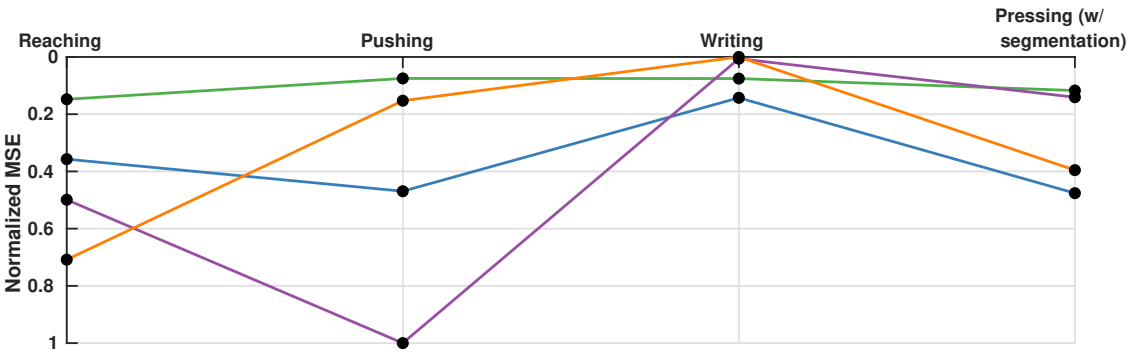


Figure 3.7: Normalized mean squared error for different algorithms across all tasks. Note that the vertical axis direction is flipped.

noise is potentially more important than having specific experience with kinesthetic teaching. This insight could direct future work on developing training guidelines to quickly increase novices' expertise. Additionally, an extension may study whether providing supplementary directions (e.g. about speed, waypoints, and direction of motion) to novice users beyond the baseline instruction improves overall performance.

3.5 Quantitative Metric Evaluations

While the evaluations reported thus far are based on a qualitative measure of performance, existing LfD literature employs quantitative metrics for this purpose. One such metric is the mean squared error (MSE) [68], which measures the deviation of reproduced motions from demonstrated trajectories. We examine whether there is a correlation between the MSE and the evaluator ratings.

We first reproduced demonstrations by querying the trained skill models from the same initial positions as the demonstrations. To account for the difference in speed between demonstrations and reproductions, we further used dynamic time warping (DTW). The MSE is then given by:

$$MSE(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{N}\frac{1}{T}\sum_{n=1}^{N}\sum_{t=0}^{T} \|\boldsymbol{x}_{t,n} - \boldsymbol{y}_{t,n}\|^2$$

where $\boldsymbol{x}_{t,n}$ and $\boldsymbol{y}_{t,n}$ are the datapoints from the demonstrated and time-aligned reproduced trajectories respectively. Furthermore, T is the length of the demonstration while N is the

number of demonstrations in the demonstration set.

Figure 3.7 reports the MSE scores, averaged over starting positions and demonstrators, plotted against the tasks. The vertical axis represents MSE scores, normalized to lie in the range 0 to 1. Note that the direction of the vertical axis for MSE scores has been reversed such that moving up the vertical axis corresponds to improvement in performance in terms of MSE. To compare against the user ratings, we view Fig. 3.7 alongside Fig. 3.5b. For each task, we ranked the approaches in terms of the MSE scores and the user ratings respectively and compared the two rankings.

Overall, despite a common assumption to the contrary, we observe that MSE is not an accurate predictor of generalization performance of a skill learning approach. This is particularly evident for the *writing* task. For this task, the AMT users were observed to care more about the shape of the executed motion as opposed to its position profile. However, MSE only measures deviations in positions from the demonstrations. Hence, while all the approaches were predicted to perform well according to MSE, only TLGC was able to draw an S-shape curve on most occasions and hence get high ratings. Furthermore, we also observe that MSE gives little information about the capability of a model to achieve the task goals. In particular, for the *pushing* task, we see that all the approaches were rated highly since they mostly achieved the goal of closing the box towards the end of execution. The users were observed to care less about the trajectory while approaching the box. However, MSE considers the entire length of the trajectories, therefore approaches that fit the data better received higher scores.

3.6 Conclusions and Discussion

We have presented a large-scale evaluation of four skill learning approaches across four real-world tasks. Our conclusions are based on 720 robot task executions and 3600 ratings provided by AMT users who evaluated the robot trajectories in terms of safety, efficiency, and success in achieving the goals of the task.

Here, we share algorithm-specific observations to guide users in selecting the appropriate method for their use case.

3.6.1 Algorithmic Observations

For those planning to use a dynamics-based approach such as CLFDM, it may be useful to note that while such methods guarantee reaching a target location, they cannot ensure *safety* or *efficiency* of executions. However, both these factors have great real-world significance, as noted by evaluators who rated CLFDM on *reaching* and *writing*. CLFDM is also more sensitive to changes in distance from the target. Segmenting the task can mitigate this problem to a certain extent.

Time-parametrized approaches, e.g. ProMP and TpGMM, can be suitable on tasks which impose a strong end-position constraint but little constraint on direction-of-motion or shape of motion (e.g., *pushing* and *reaching*). However, starting very close to the goal can immensely affect performance. This is because time-parametrized approaches in general are not robust to large spatio-temporal perturbations. One should take care to ensure the robot does not start too close to the final position unless a majority of the provided demonstrations are in the vicinity of this desired starting position.

For tasks with strong constraints in shape, a geometric approach like TLGC can be more suitable. We conclude this by observing the consistency of TLGC's performance on the *writing* and *pressing* tasks. This is perhaps because TLGC explicitly encodes the shape of the demonstrated motions and minimizes deviations from this shape during reproduction.

3.6.2 Research Insights

This subsection provides general, algorithm-independent research insights learned from this benchmarking effort. We hope this knowledge will guide researchers in developing more robust techniques.

- Approaches with different model representation perform differently over tasks with

various constraints. Our evaluations suggest that none of the approaches worked well across every task. While TLGC, the approach with a geometric representation, worked well for tasks with strong constraints on the shape of motions (e.g., *writing*), ProMP, with a time-parametrized probabilistic representation, was found to be most consistent on tasks with positional (e.g., goal location) constraints.

- Generalization quality decreases as the new starting positions go farther from the original starting positions. None of the approaches were able to consistently generalize to such starting positions in an acceptable manner.
- Task complexity affects the approaches' generalization ability. Our results show that algorithms generalize better for tasks with simpler constraints, usually struggling on tasks with directional and positional constraints.
- For long-horizon tasks with multiple position constraints (e.g. via-points) alongside constraints on the direction of motion, motion segmentation can be beneficial.
- Higher user experience level positively impacts the performance of the approaches. Our findings also show that algorithm performance is affected by the *quality* of demonstrations from users with varying experience.
- Conventional metrics may not be good predictors of performance. We have found that the quantitative MSE does not reliably predict performance across many tasks.

Part II

Learning Time-Dependent Skill Representations

CHAPTER 4

MULTI-COORDINATE COST BALANCING

One of the skill learning representations we propose in this dissertation employs multiple differential coordinates to encode geometric properties of motions. As discussed and evaluated in previous chapters, existing works in LfD have contributed a wide range of mathematical representations that encode skills from human demonstrations, ranging from Gaussian mixture models (GMMs) [8, 9, 16] to neural networks [60, 78]. Each of these representations is used to encode the demonstrations in a predefined space or coordinate system (e.g., Cartesian coordinates). In other words, a single best coordinate system for any given skill is assumed to both exist and be known. However, as we show in this work, the assumption that a single best coordinate system exists for each task does not hold. Further, encoding in only a single coordinate system prohibits the model from capturing some of the geometric features that underly a demonstrated skill.

We contribute a learning framework that encodes demonstrations simultaneously in multiple coordinates, and balances the relative influences of the learned models in generating reproductions. The proposed framework, named Multi-Coordinate Cost Balancing (MCCB), encodes demonstrations in three differential coordinates: Cartesian, tangent, and Laplacian (Section 4.1.1). Simultaneously learning in these three coordinates allows our method to capture all of the underlying geometric properties that are central to a given skill. MCCB encodes the joint density of the time index and the demonstrations in each differential coordinate frame using a separate statistical model. Thus, given any time instant, we are able to readily obtain the conditional mean and covariance in each coordinate system (Section 4.1.2). MCCB generates reproductions by solving an optimization problem with a blended cost function that consists of one term per coordinate. Each term penalizes deviations from the norm, weighted by the inverse of the expected variance in the corresponding coordinate

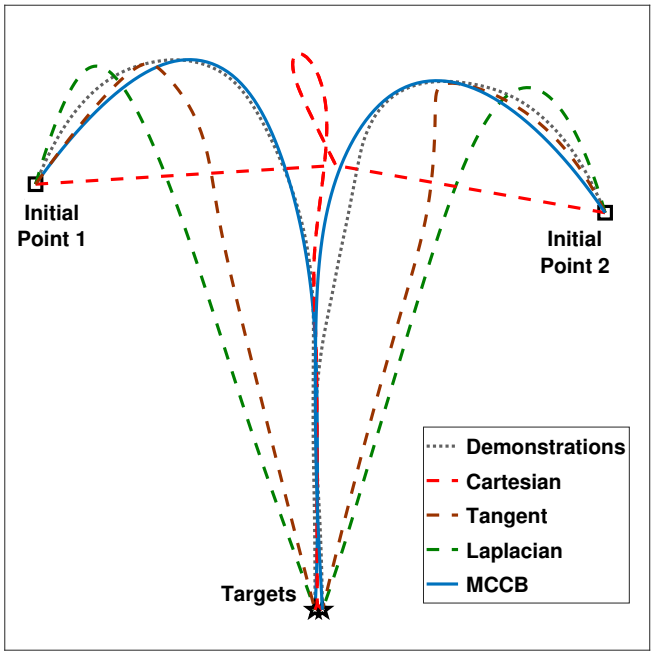


Figure 4.1: A comparison of reproductions generated by considering different coordinates, illustrating the need for cost balancing.

system (Section 4.1.3). Further, we subject the optimization problem to linear constraints on the reproductions, such as initial, target, and via point constraints. Our constrained optimization problem is convex with respect to the reproduction and hence can be solved efficiently.

A major hurdle in learning a wide variety of skills, without significant parameter tweaking, is that the relative importance of each differential coordinate (or the geometric feature) in encoding a given task is unknown ahead of time. For instance, consider the problem of encoding the demonstrations illustrated in Fig. 4.1. Using any one coordinate system in isolation, even when the most suitable one is known, does not yield good reproductions (the red, brown, and green dashed lines). To alleviate this problem, MCCB preferentially weights the costs defined in each coordinate (Fig. 4.2). Importantly, MCCB learns the optimal weights directly from the demonstrations without making task-dependent assumptions. To this end, MCCB solves a meta optimization problem that aims to minimize reproduction errors (Section 4.1.4). As shown by the solid blue lines in Fig. 4.1, a cost function that optimally

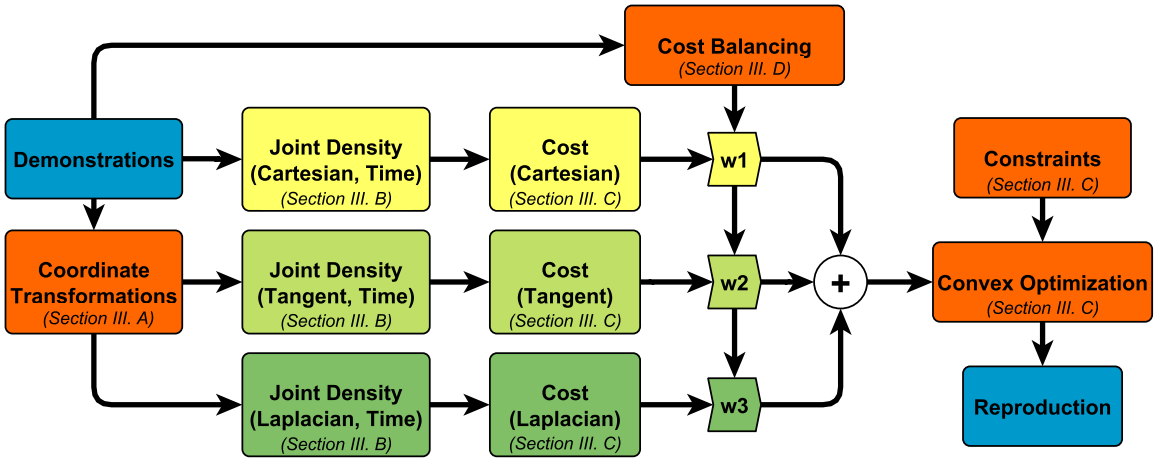


Figure 4.2: A flow diagram illustrating MCCB.

balances the costs in each coordinate yields better reproductions than any single-coordinate method.

In summary, we contribute a unified task-independent learning framework that (1) encodes demonstrations simultaneously in multiple differential coordinates, (2) defines a blended cost function that incentivizes conformance to the norm in each coordinate system while considering expected variance, and (3) learns optimal weights directly from the demonstrations to balance the relative influence of each differential coordinate in generating reproductions. Further, MCCB is compatible with and complementary to several existing LfD methods that utilize different statistical representations and coordinate systems [45, 12, 69, 79, 80, 46, 32].

4.1 Methodology

This section describes the technical details of MCCB and its work flow as illustrated in Fig. 4.2. First, we describe the differential coordinates employed in MCCB. Second, we detail our method for encoding motion constraints in each of the coordinates in terms of a cost function. Third, we present our constrained optimization framework to reproduce the skill. Lastly, we introduce our meta-optimization method for learning the correct combination of cost functions employed in the skill reproduction routine.

4.1.1 Differential Coordinate Transformations

The differential coordinates and their corresponding transformations used in MCCB are as follows.

Cartesian: Let a discrete finite-length trajectory in n-dimensional *Cartesian coordinates* be denoted by $\boldsymbol{X} = [x(1)\ x(2) \cdots x(T)]^\top \in \mathbb{R}^{T \times n}$ and let $x(t) \in \mathbb{R}^n$ denote a discrete sample at time index t. This trajectory can be represented using a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of vertices representing the samples in the trajectory and $\mathcal{E}$ is the set of edges that represent the connections between the samples in the trajectory. The neighborhood $\mathcal{N}_t$ of each vertex $\mathcal{V}_t$ is defined by the set of adjacent vertices $\mathcal{V}'_t$. In the case of discrete-time trajectories, the edges between any given vertex and its two neighbors are assumed to carry unit weights, while all other edges carry zero weights.

Laplacian: It is known that the discrete Laplace-Beltrami operator for the trajectory $\boldsymbol{X}$ provides the *Laplacian coordinate* $\delta(t)$ as $\delta(t) \triangleq \sum_{t' \in \mathcal{N}_t} \frac{1}{\sum_{t' \in \mathcal{N}_t} 1}(x(t) - x(t'))$ [81]. Note that the above relationship can be written as a linear differential operator in matrix form

$$\boldsymbol{\Delta} = \boldsymbol{L}\boldsymbol{X} \tag{4.1}$$

where $\boldsymbol{\Delta} = [\delta(1)\ \delta(2) \cdots \delta(T)]^\top \in \mathbb{R}^{T \times n}$ is the trajectory in the Laplacian coordinates, and $\boldsymbol{L} \in \mathbb{R}^{T \times T}$, called the graph Laplacian, is given by

$$\boldsymbol{L} = \begin{bmatrix} 1 & -1 & 0 & \dots & \dots & 0 \\ -0.5 & 1 & -0.5 & 0 & \dots & 0 \\ 0 & -0.5 & 1 & -0.5 & \dots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & -0.5 & 1 & -0.5 \\ 0 & \dots & \dots & 0 & -1 & 1 \end{bmatrix} \tag{4.2}$$

As pointed out in [32], the Laplacian coordinates have meaningful geometric interpretations. Specifically, the Laplacian coordinates can be seen as the discrete approximations of the derivative of the unit tangent vectors of an arc-length parametrized continuous trajectory. In other words, the Laplacian coordinates measure the deviation of each sample from the centroid of its neighbors.

Tangent: While the Laplacian coordinates are discrete approximations of second order differential transformations, a discrete approximation of the first differential transformation is possible. Consider such a first order transformation using first order finite differences defined as $\gamma(t) \triangleq (x(t+1) - x(t))$, where $\gamma(t)$ is called the *tangent coordinate*. The matrix form of the above relationship results in a linear differential operator given by

$$\boldsymbol{\Gamma} = \boldsymbol{G}\boldsymbol{X} \tag{4.3}$$

where $\boldsymbol{\Gamma} = [\gamma(1)\ \gamma(2) \cdots \gamma(T)]^\top \in \mathbb{R}^{T \times n}$ is the trajectory in the tangent coordinates and $\boldsymbol{G} \in \mathbb{R}^{T \times T}$, called the graph incidence matrix, is given by

$$\boldsymbol{G} = \begin{bmatrix} -1 & 1 & 0 & \dots & \dots & 0 \\ 0 & -1 & 1 & 0 & \dots & 0 \\ 0 & 0 & -1 & 1 & \dots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & 0 & -1 & 1 \\ 0 & \dots & \dots & 0 & 0 & -1 \end{bmatrix} \tag{4.4}$$

Similar to the Laplacian coordinates, the tangent coordinates have geometric interpretations. Specifically, the tangent coordinates can be seen as discrete approximations of the unnormalized tangent vectors of an arc-length parametrized continuous trajectory, i.e., the tangent coordinates measure the local direction of motion at each sample of the trajectory.

In our work, we assume that a set of N demonstrations in the Cartesian coordinates are available. Let the jth demonstration be denoted by $\boldsymbol{X}_d^j = [x_d^j(1)\ x_d^j(2) \cdots x_d^j(T)]^\top \in \mathbb{R}^{T \times n}$. Note that if the raw demonstrations are of varying duration in time, we perform time alignment using dynamic time warping. MCCB transforms each obtained demonstration $\boldsymbol{X}_d^j$ into a trajectory in the tangent coordinates (denoted by $\boldsymbol{\Gamma}_d^j$) and a trajectory in Laplacian coordinates (denoted by $\boldsymbol{\Delta}_d^j$) using (4.1) and (4.3), respectively.

4.1.2 Encoding in Multiple Differential Coordinates

This subsection defines the costs associated with each coordinate. With the demonstrations available in all three differential coordinates, we employ three independent Gaussian mixture

models (GMMs)[1] to approximate the joint probability densities of time and the samples in each coordinate system.

The GMM associated with the Cartesian coordinates attempts to approximate the joint density of t and x, i.e. $\mathcal{P}(t, x; \theta_C) = \sum_{k=1}^{K_C} \mathcal{P}(k)\mathcal{P}(t, x|k)$, where K_C is the number of Gaussian basis functions, $\mathcal{P}(k) = \pi_C^k$ is the prior associated with the kth basis function, $\theta_C = \{\mu_C^1 \cdots \mu_C^{K_C}, \Sigma_C^1 \cdots \Sigma_C^{K_C}, \pi_C^1 \cdots \pi_C^{K_C}\}$ is the set of parameters of the GMM. Furthermore, the conditional density of the kth Gaussian basis function parameterized by it mean μ_C^k and covariance Σ_C^k, is given by,

$$\mathcal{P}(t, x|k) \sim \mathcal{N}\left(\begin{bmatrix} t \\ x \end{bmatrix}; \mu_C^k, \Sigma_C^k\right), \qquad \text{where, } \mu_C^k = \begin{bmatrix} \mu_t^k \\ \mu_x^k \end{bmatrix}, \text{and } \Sigma_C^k = \begin{bmatrix} \Sigma_t^k & \Sigma_{t,x}^k \\ \Sigma_{x,t}^k & \Sigma_x^k \end{bmatrix}. \tag{4.5}$$

We learn the parameters θ_C of the model using the Expectation-Maximization algorithm based on the demonstrations $\{\boldsymbol{X}_d^j\}_{j=1}^N$. Given the learned model and a time instant, the expected value of the conditional density $\mathcal{P}(x|t)$ is given by Gaussian mixture regression (GMR) [82] as follows

$$\hat{x}(t) = \mathbb{E}[x|t] = \sum_{k=1}^{K_C} h_C^k(t)(A_C^k t + b_C^k) \tag{4.6}$$

where $h_C^k(t) = \frac{\mathcal{P}(k)\mathcal{P}(t|k)}{\sum_{i=1}^{K_C} \mathcal{P}(i)\mathcal{P}(t|i)}$, $A_C^k = \Sigma_{x,t}^k(\Sigma_t^k)^{-1}$, $b^k = \mu_x^k + (t - \mu_t^k)$, and the conditional covariance is given by

$$\hat{\Sigma}_x(t) = Var[x|t] = \sum_{k=1}^{K_C} {h_C^k}^2 \, (\Sigma_x^k - \Sigma_{x,t}^k(\Sigma_t^k)^{-1}\Sigma_{t,x}) \tag{4.7}$$

Similar to the GMM learned in the Cartesian coordinates, we learn a second GMM in the tangent coordinates based on the demonstrations $\{\boldsymbol{\Gamma}_d^j\}_{j=1}^N$, and a third GMM in the Laplacian coordinates based on the demonstrations $\{\boldsymbol{\Delta}_d^j\}_{j=1}^N$. The expected values of the

[1] MCCB does not rely on the use of GMMs and any statistical representation that can provide the conditional estimates will suffice.

conditional densities $\mathcal{P}(\gamma|t)$ and $\mathcal{P}(\delta|t)$ are given by

$$\hat{\gamma}(t) = \mathbb{E}[\gamma|t] = \sum_{k=1}^{K_G} h_G^k(t)(A_G^k t + b_G^k) \tag{4.8}$$

$$\hat{\delta}(t) = \mathbb{E}[\delta|t] = \sum_{k=1}^{K_L} h_L^k(t)(A_L^k t + b_L^k) \tag{4.9}$$

and the corresponding conditional expectations are given by

$$\hat{\Sigma}_\gamma(t) = \mathrm{Var}[\gamma|t] = \sum_{k=1}^{K_G} (h_G^k)^2 \, (\Sigma_\gamma^k - \Sigma_{\gamma,t}^k (\Sigma_t^k)^{-1} \Sigma_{t,\gamma}) \tag{4.10}$$

$$\hat{\Sigma}_\delta(t) = \mathrm{Var}[\delta|t] = \sum_{k=1}^{K_L} (h_L^k)^2 \, (\Sigma_\delta^k - \Sigma_{\delta,t}^k (\Sigma_t^k)^{-1} \Sigma_{t,\delta}) \tag{4.11}$$

where the variables in (4.8)-(4.11) with subscripts G and L correspond to the tangent and Laplacian coordinates, respectively, and are defined similarly to the ones in (4.6)-(4.7).

4.1.3 Imitation via Optimization

In this section, we explain the design of our multi-coordinate cost function. MCCB generates reproductions by solving a constrained optimization problem given by

$$\begin{aligned} \boldsymbol{X}_r = \arg\min_{\boldsymbol{X}} \; & w_C J_C(\boldsymbol{X}) + w_G J_G(\boldsymbol{X}) \\ & + w_L J_L(\boldsymbol{X}) \end{aligned} \tag{4.12}$$

$$\text{s.t.} \qquad P_x \boldsymbol{X} = \boldsymbol{X}^* \tag{4.13}$$

where $\boldsymbol{X}_r \in \mathbb{R}^{T \times n}$ is the reproduction, $w_C,\ w_G,\ w_L \in \mathbb{R}^+$ are positive weights; $J_C,\ J_G,\ J_L : \mathbb{R}^{T \times n} \to \mathbb{R}^+$ are cost functions in the Cartesian, tangent, and Laplacian coordinates, respectively; $P_x \in \mathbb{R}^{m \times T}$ and $\boldsymbol{X}^* \in \mathbb{R}^{m \times n}$ define $m \in \mathbb{Z}^+$ linear constraints on $\boldsymbol{X}_r$. In practice, $m << n$ and we use the linear constraints to enforce constraints on initial, target, and via points.

We define the cost function in each coordinate system as follows

$$J_C(\boldsymbol{X}) = (\boldsymbol{X}(:) - \hat{\boldsymbol{X}}(:))^\top (\hat{\boldsymbol{\Sigma}}_{\boldsymbol{X}})^{-1} (\boldsymbol{X}(:) - \hat{\boldsymbol{X}}(:)) \tag{4.14}$$

$$J_G(\boldsymbol{X}) = (\boldsymbol{\Gamma}(:) - \hat{\boldsymbol{\Gamma}}(:))^\top (\hat{\boldsymbol{\Sigma}}_{\boldsymbol{\Gamma}})^{-1} (\boldsymbol{\Gamma}(:) - \hat{\boldsymbol{\Gamma}}(:)) \tag{4.15}$$

$$J_L(\boldsymbol{X}) = (\boldsymbol{\Delta}(:) - \hat{\boldsymbol{\Delta}}(:))^\top (\hat{\boldsymbol{\Sigma}}_{\boldsymbol{\Delta}})^{-1} (\boldsymbol{\Delta}(:) - \hat{\boldsymbol{\Delta}}(:)) \tag{4.16}$$

where $\hat{\boldsymbol{\Sigma}}_{\boldsymbol{X}}, \hat{\boldsymbol{\Sigma}}_{\boldsymbol{\Gamma}}, \hat{\boldsymbol{\Sigma}}_{\boldsymbol{\Delta}} \in \mathbb{R}^{\boldsymbol{nT} \times \boldsymbol{nT}}$ denote the block diagonal matrices formed with the conditional covariances $\hat{\Sigma}_x(t), \hat{\Sigma}_\gamma(t)$, and $\hat{\Sigma}_\delta(t)$, respectively, for all values of t. Further, the notation $(:)$ denotes vectorization - for instance, $\boldsymbol{X}(:), \hat{\boldsymbol{X}}(:) \in \mathbb{R}^{nT}$ denote the vectorized trajectories formed by vertically stacking $x(t)$ and $\hat{x}(t)$ for all values of t, respectively. Note that we construct the trajectories Γ and Δ in (4.15) and (4.16) from $\boldsymbol{X}$ via the linear operators defined in (4.3) and (4.1), respectively. MCCB penalizes deviations from the conditional mean in each coordinate system. However, deviations are penalized less (more) severely if high (low) variance is observed in the demonstrations at any given time.

4.1.4 Automated Cost Balancing

In order to obtain reproductions that successfully imitate demonstrations of a wide variety of skills, the weights $w_C,\ w_G$, and w_L have to be chosen with care. Indeed, they preferentially weight the costs defined in each differential coordinate and thereby manipulate the relative incentive for successful imitation in each coordinate system.

We learn these weights directly from the available demonstrations. Note that, for known weights, the constrained optimization problem in (4.12) is convex in $\boldsymbol{X}$. We estimate the weights in the following form

$$\hat{w}_C = \frac{\alpha_C}{\beta_C}; \quad \hat{w}_G = \frac{\alpha_G}{\beta_G}; \quad \hat{w}_L = \frac{\alpha_L}{\beta_L} \tag{4.17}$$

where $\beta_C,\ \beta_G,\ \beta_L \in (0, 1]$, such that $\sum_i \beta_i = 1$, are positive scaling factors used to correct for inherent differences in the magnitudes of the costs, and $\alpha_C,\ \alpha_G,\ \alpha_L \in [0, 1]$, such

that $\sum_i \alpha_i = 1$, are positive weights used to preferentially weight the cost defined in each coordinate system. MCCB estimates the scaling factors β_i's as follows

$$\beta_i = \frac{\sum_{j=1}^{N} J_i(\boldsymbol{X}_d^j)}{\sum_l \sum_{j=1}^{N} J_l(\boldsymbol{X}_d^j)}, \quad \forall i, l = \{C, G, L\} \tag{4.18}$$

With the scaling factors compensating the inherent scale difference in the costs, we compute the preferential weighting factors α_i's that minimize reproduction error. To this end, we formulate the following meta optimization problem

$$\{\alpha_C, \alpha_G, \alpha_L\} = \arg \min_{\alpha_C, \alpha_G, \alpha_L} \sum_{j=1}^{N} \mathrm{SSE}(\boldsymbol{X}_r^j, \boldsymbol{X}_d^j) \tag{4.19}$$

$$\text{s.t.} \sum_i \alpha_i = 1, \ \forall i = \{C, G, L\} \tag{4.20}$$

where $\mathrm{SSE}(\cdot)$ denotes the sum of squared errors computed over time, and $\boldsymbol{X}_r^j$ is the solution to the following optimization problem

$$\boldsymbol{X}_r^j = \arg \min_{\boldsymbol{X}} \left(\frac{\alpha_C}{\beta_C}\right) J_C(\boldsymbol{X}) + \left(\frac{\alpha_G}{\beta_G}\right) J_G(\boldsymbol{X}) + \left(\frac{\alpha_L}{\beta_L}\right) J_L(\boldsymbol{X}) \tag{4.21}$$

$$\text{s.t.} \qquad P_x \boldsymbol{X} = \boldsymbol{X}_j^* \tag{4.22}$$

where $P_x \boldsymbol{X} = \boldsymbol{X}_j^*$ denotes specific linear constraints pertaining to the demonstration $\boldsymbol{X}_d^j$, such as initial, target, and via points. Solving the above meta-optimization problem results in the preferential weights α_i's that minimize reproduction errors of the solutions generated by the original constrained optimization problem in (4.12)-(4.13).

4.2 Experimental Evaluation

This section describes the design and discusses the results of four experiments conducted to evaluate MCCB[2]. In each experiment, we compared the performances of the following approaches:

1. *Cartesian-coordinates*: $w_C = 1$, $w_G = 0$, $w_L = 0$
2. *Tangent-coordinates*: $w_C = 0$, $w_G = 1$, $w_L = 0$
3. *Laplacian-coordinates*: $w_C = 0$, $w_G = 0$, $w_L = 1$
4. *Uniform weighting*: $w_C = 1/3$, $w_G = 1/3$, $w_L = 1/3$
5. *MCCB*: $w_C = \hat{w_C}$, $w_G = \hat{w_G}$, $w_L = \hat{w_L}$

We measured the performance of each approach by the following geometric and kinematic metrics: *Swept Error Area (SEA)* [15], *Sum of Squared Errors (SSE)*, *Dynamic Time Warping Distance (DTWD)*, and *Frechet Distance (FD)* [83]. These metrics allow us to evaluate different aspects of each method's performance. The SEA and SSE metrics penalize both spatial and temporal misalignment, and thus evaluate kinematic performance. On the other hand, the DTWD and FD metrics penalize spatial misalignment while disregarding time misalignment, and thus evaluate geometric performance. Further, the SEA, SSE, and DTWD metrics evaluate aggregate performance by summing over or averaging across all the samples of each reproduction. The FD metric, on the other hand, computes the shortest possible cord length required to connect the demonstration and the reproduction in space while allowing time re-parametrization of either trajectory, and thus measures maximal deviation in space. Note that the SEA metric is restricted to 2-dimensional data, so we only report it for one of our experiments.

In all the experiments, we used the position constraints in (4.13) to enforce both initial and end point constraints uniformly across all the methods being compared. Further, we

[2]Video of experiments: https://youtu.be/bVlK1f8e0Dg

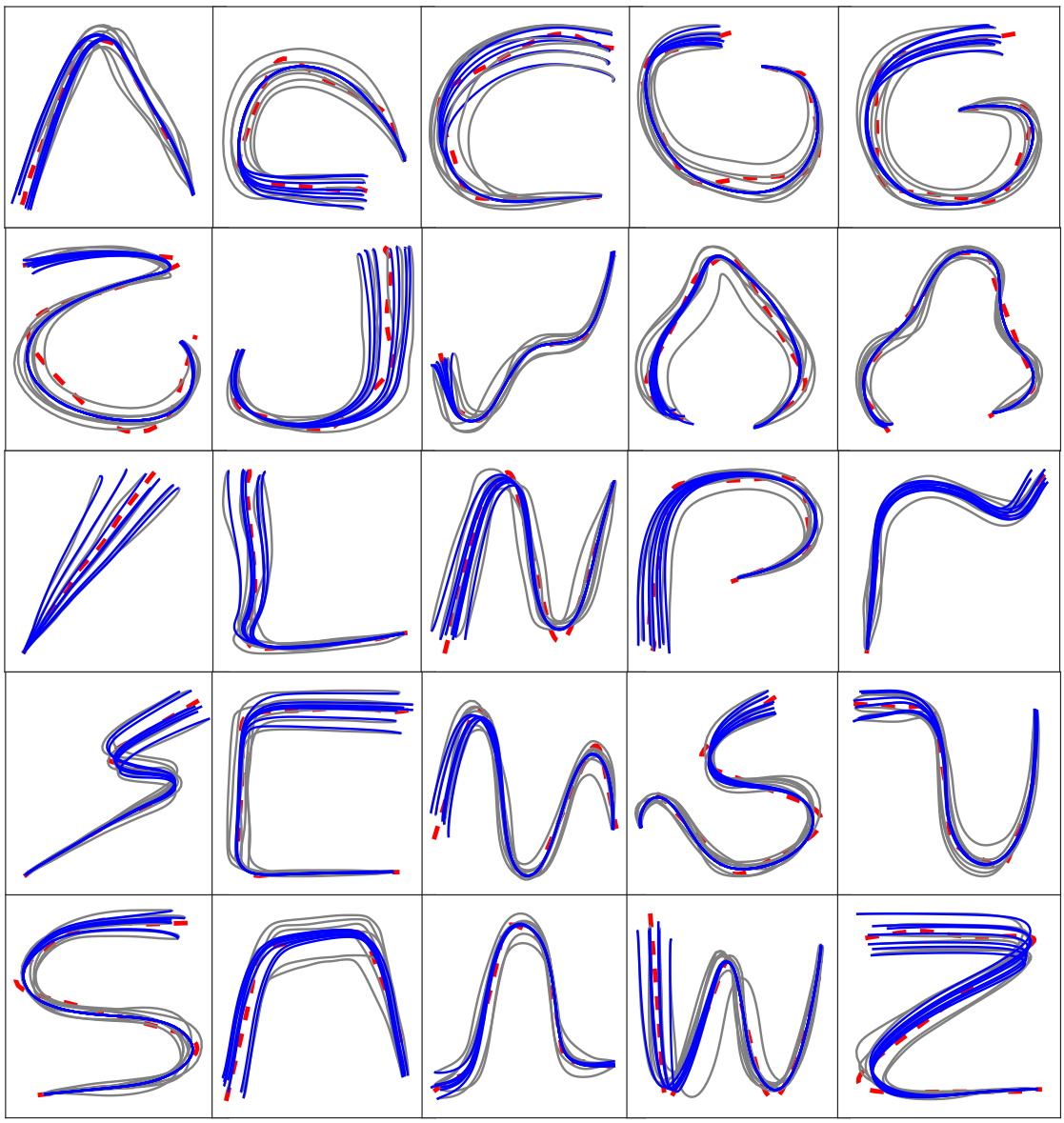


Figure 4.3: Qualitative performance of MCCB on the LASA handwriting dataset. Demonstration (gray), reproductions (blue), and expected mean position (dashed red) are shown.

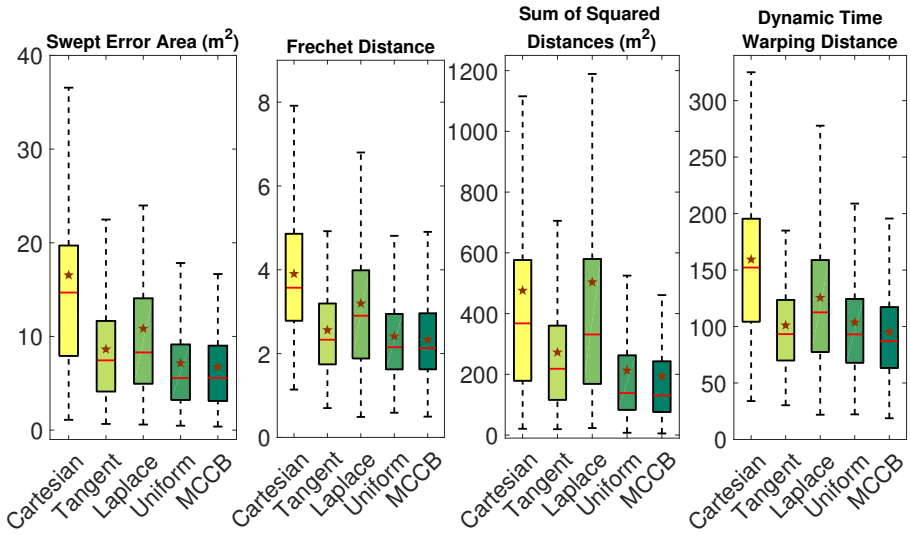


Figure 4.4: Box plots, with mean (brown star) and median (red line), illustrate the performance of each approach on the handwriting task.

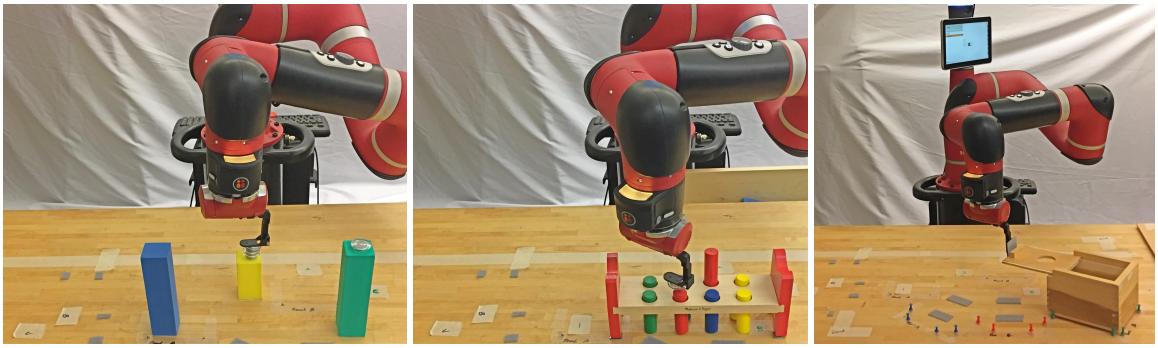


Figure 4.5: Snapshots illustrating the experimental setup for the picking (left), pressing (center), and pushing (right) skills.

uniformly set the number of Gaussian basis functions to five across all the coordinates and all the experiments.

4.2.1 Handwriting Skill

This experiment evaluates MCCB on the publicly available LASA human handwriting library [9], that consists of handwriting motions collected from pen input using a Tablet PC. The library contains a total of 25 handwriting motions, each with 7 demonstrations.

Fig. 4.3 shows that MCCB yields reproductions that are qualitatively similar to the demonstrations while satisfying the end-point constraints across all motions. As shown in

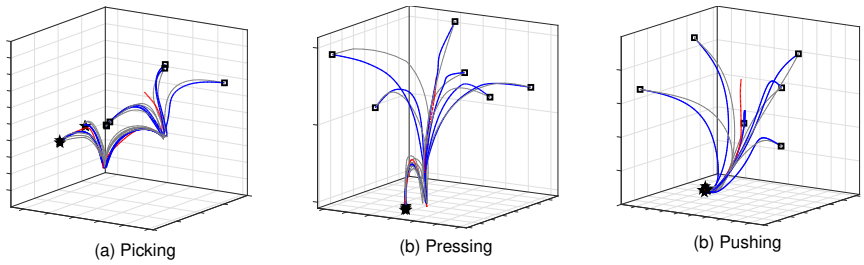


Figure 4.6: Qualitative performance of MCCB on the picking, pressing, and pushing datasets. Demonstration (gray), reproductions (blue), expected mean position (dashed red), initial (black squares), and target (black stars) are shown.

Fig. 4.4, quantitative analysis indicates that MCCB ($\bar{\alpha}_C = 0.1814,\ \bar{\alpha}_G = 0.4958,\ \bar{\alpha}_L = 0.3228$)[3] and three of the four baselines performed comparably with respect to the SEA, FD, SSE, and DTWD metrics, while the Cartesian baseline performed poorly in comparison. This is consistent with the fact that the demonstrations within the LASA dataset emphasize strong similarities in shape.

4.2.2 Picking Skill

The second experiment evaluates the performance of MCCB in a picking task (Fig. 4.5). The data consists of six kinesthetic demonstrations, each a 3-dimensional robot end-effector position trajectory recorded as a human guided the robot in picking up two magnets atop two blocks. We enforced two via-point constraints (one at each picking point) in addition to the end-point constraints.

As shown in Fig. 4.6(a), MCCB generated reproductions that are qualitatively similar to the demonstrations while satisfying all the position constraints. Quantitative evaluations reveal that learning in tangent coordinates yielded better reproductions than learning in Cartesian and Laplacian coordinates (Fig. 4.7). This was expected since the demonstrations of this task, much like the LASA dataset, emphasize shape similarity. Further, MCCB ($\alpha_C = 0.2362,\ \alpha_G = 0.5451,\ \alpha_L = 0.2187$) yielded the best performance, with respect to all three metrics. In fact, uniform weighting yielded poorer results, with respect to all three

[3] Weighting factors averaged over all 25 skills in the LASA dataset

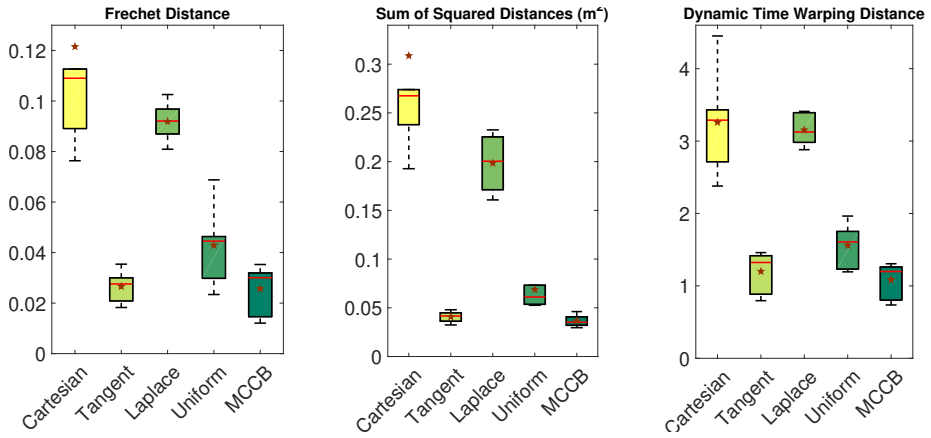


Figure 4.7: Box plots, with mean (brown star) and median (red line), illustrate the performance of each approach on the picking dataset.

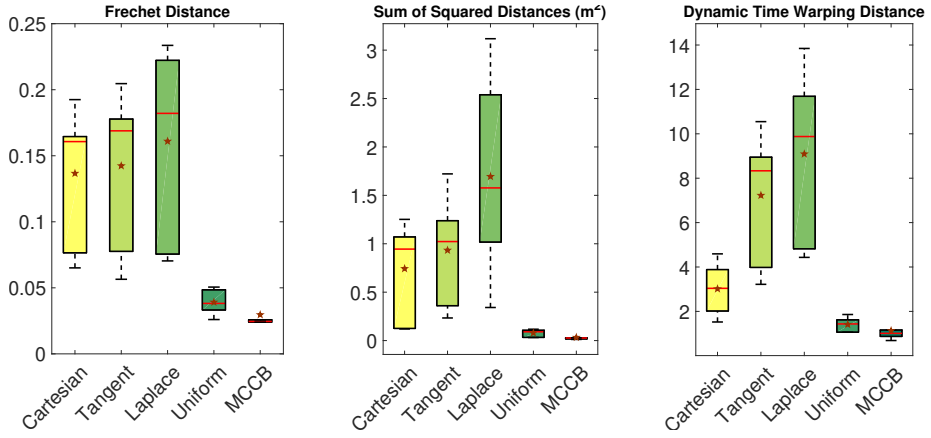


Figure 4.8: Box plots, with mean (brown star) and median (red line), illustrate the performance of each approach on the pressing dataset.

metrics, than when considering only the tangent coordinates. The results of this experiment show that while multi-coordinate methods can yield strong performance, it is critical that we balance the weights appropriately.

4.2.3 Pressing Skill

In this experiment, we evaluated MCCB's ability to learn pressing skills (Fig. 4.5). The data consists of six kinesthetic demonstrations, each a 3-dimensional robot end-effector position trajectory recorded as a human guided the robot in pressing two cylindrical pegs into their respective holes.

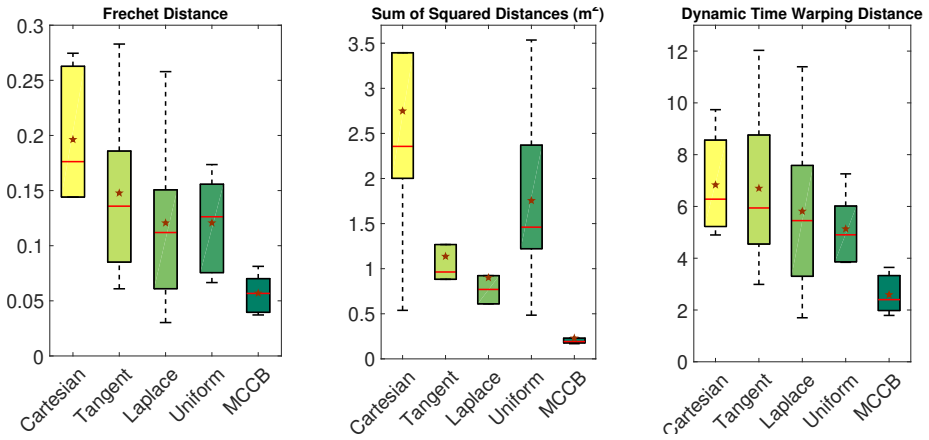


Figure 4.9: Box plots, with mean (brown star) and median (red line), illustrate the performance of each approach on the pushing dataset.

As shown in Fig. 4.6(b), MCCB successfully reproduced the demonstrations. Note that MCCB is capable of automatically capturing and reproducing the consistencies across the demonstrations in certain regions without any position constraints. Fig. 4.8 illustrates the performance of MCCB and the baselines with respect to three different metrics. Learning in Cartesian coordinates resulted in the better performance compared to learning in tangent and Laplacian coordinates. Quantitative evaluations further demonstrate that MCCB ($\alpha_C = 0.6735,\ \alpha_G = 0.2034,\ \alpha_L = 0.1231$) consistently yielded the best performance with respect to all three metrics. The results of this experiment, in light of the results in Section 4.2.2, suggest that the relative importance of each of the differential coordinates vary across different skills.

4.2.4 Pushing Skill

The final experiment evaluates the performance of MCCB in a pushing task (Fig. 4.5). The data consists of six kinesthetic demonstrations, each a 3-dimensional robot end-effector position trajectory recorded as a human guided the robot in sliding closed the lid of a wooden box.

As shown in Fig. 4.6(c), MCCB successfully generated reproductions that are similar to the demonstrations. As evidenced by quantitative evaluations in Fig. 4.9, encoding

Table 4.1: The most relevant and best performing methods on each task. Orange check marks denote the most relevant coordinate and green check marks denote the best performing method.

	Single Coordinate			Multi-Coordinate	
	Cartesian	Tangent	Laplacian	Uniform W.	MCCB
Handwriting		✓ ✓	✓ ✓	✓	✓
Picking		✓			✓
Pressing	✓				✓
Pushing			✓		✓

demonstrations in the Laplacian coordinates yielded better performance, with respect to all three metrics, when compared to learning only in either of the other two coordinates, while, MCCB ($\alpha_C = 0.0123,\ \alpha_G = 0.045,\ \alpha_L = 0.9427$) consistently outperformed all the other approaches. Note that learning in the Laplacian coordinates alone resulted in better performance than uniformly weighting of all the coordinates. These results are consistent with the results from the previous sections and indicate that MCCB yields consistently good performance. The results are summarized in Table 4.1.

4.3 Discussion and Conclusion

We introduced MCCB, a learning framework for encoding demonstrations in multiple differential coordinates, and automated balancing of costs defined in those coordinates. As shown in Table 4.1, we demonstrated that the relative effectiveness of each coordinate system is not consistent across a variety of tasks since any given skill might be better suited for learning in one (or more) coordinate system(s). Furthermore, uniform weighting of costs in different coordinates does not consistently yield the best results across different skills. Indeed, uniform weighting, in some cases, yielded poorer performances compared to when only one coordinate system was used. On the other hand, MCCB learned to balance the costs and consistently yielded the best performance. Since the weights are learned directly from the demonstrations, MCCB makes no task-specific assumptions and does not require

tedious parameter tuning. Note that although we used GMMs as the base representation in this work, MCCB is agnostic to the statistical model used to encode the demonstrations in each coordinate system, and thus can be combined with other techniques, such as [45, 12, 69, 79, 80, 46, 32]. Furthermore, MCCB can be extended to include more coordinate systems that capture additional trajectory features.

CHAPTER 5

PROBABILISTIC INFERENCE WITH STRUCTURED PRIORS: COMBINED LFD AND MOTION PLANNING

The other challenge we address in this thesis involves learning *optimal* yet *feasible* robot motions. Majority of existing skill learning methods make certain assumptions regarding the feasibility of reproduced motions. Specifically, if a skill model generates task space (or equivalently workspace) motions, an inverse kinematics solver or motion planner is often employed post-hoc to generate corresponding configuration space commands. This two-step process assumes that the configuration space motions, when mapped back to the task space, would exactly coincide with the motions given by the skill model. However, this assumption breaks when a desired task space trajectory is unreachable either due to the robot's kinematic constraints, or due to the presence of obstacles in the path. It is desirable to have a method that accounts for the inter-dependence between the configuration space and task space motions, while also simultaneously reasoning about additional costs or constraints that may exist (e.g those associated with obstacle avoidance.)

We view the motion generation problem associated with a learned skill, as equivalent to finding motions that are *optimal* as per human demonstrations, but also *feasible* in terms of robot's kinematic constraints and environmental constraints. In lieu of this, similar to recent work in motion planning [84], we carry out motion generation via probabilistic inference. Specifically, to generate a new motion, we find a posterior by conditioning a *prior* over motions, encoding optimality, on a collection of *likelihood* functions that characterize feasibility. Unlike motion planning, the prior is learned from human demonstrations and is parameterized by a time-dependent stochastic dynamical system. The resulting algorithm, combined learning from demonstration and motion planning (CLAMP), further utilizes efficient probabilistic inference tools [85] to generate optimal and feasible motions.

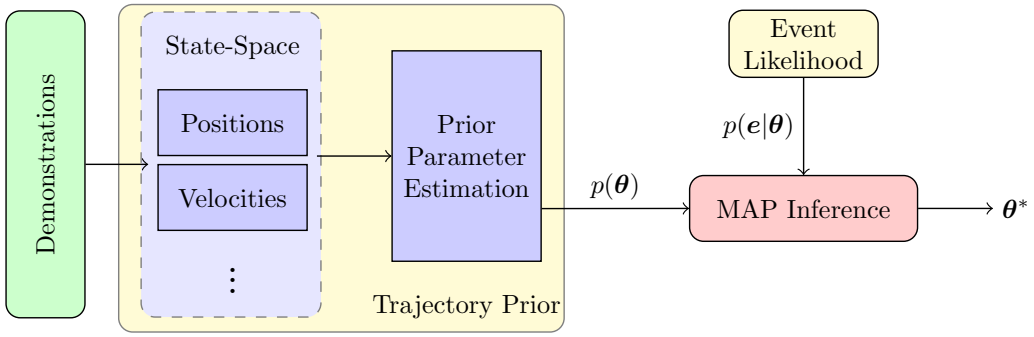


Figure 5.1: Overview of CLAMP. A trajectory prior is learned from human demonstrations, which is employed to generate new motions via probabilistic inference.

Our specific contributions with CLAMP include: (i) a skill model (trajectory prior) that encodes the spatio-temporal skill constraints from demonstrations, and enables efficient inference; and (ii) a motion generation framework that utilizes the prior and finds *optimal* and *feasible* robot motions via efficient probabilistic inference. To validate CLAMP, we learn three skills on a JACO^2 6-DOF robot arm: *box-opening*, *drawer-opening*, and *picking*. Our results show that CLAMP is capable of successfully executing the aforementioned skills from new initial robot states in the presence of obstacles in the environment.

5.1 Motion Generation via Probabilistic Inference

A robot trajectory is given by a continuous-time function mapping time t to the instantaneous robot state $\boldsymbol{\theta}(t) \in \mathbb{R}^d$. At any stage, CLAMP maintains a probability distribution over trajectories, the mode of which, i.e. *maximum a posteriori* (MAP), gives the optimal and feasible trajectory. The aforementioned probabilistic formulation naturally allows the incorporation of optimality metrics learned from demonstrations in the form of a prior distribution. Furthermore, feasibility constraints can be encoded into *likelihood* functions specified in terms of a collection of binary events $\mathbf{e}$. Figure 6.2 illustrates our proposed framework.

Trajectory Prior – We define the prior distribution over continuous-time trajectories by a vector-valued Gaussian process (GP) [86], $\boldsymbol{\theta}(t) \sim \mathcal{GP}(\boldsymbol{\mu}(t), \boldsymbol{\mathcal{K}}(t, t'))$, with mean $\boldsymbol{\mu}(t) \in \mathbb{R}^d$

and covariance $\boldsymbol{\mathcal{K}}(t, t') \in \mathbb{R}_{+}^{d \times d}$. By definition of a GP, for any finite collection of times $\{t_i\}_{i=0}^{N}$, the corresponding set of robot states $\boldsymbol{\theta} \doteq [\boldsymbol{\theta}_0, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_N]^T$ are jointly Gaussian distributed,

$$p(\boldsymbol{\theta}) \propto \exp\{-\frac{1}{2}\|\boldsymbol{\theta} - \boldsymbol{\mu}\|_{\boldsymbol{\mathcal{K}}}^2\}, \tag{5.1}$$

where

$$\boldsymbol{\mu} \doteq [\boldsymbol{\mu}(t_0), \boldsymbol{\mu}(t_1), \ldots, \boldsymbol{\mu}(t_N)]^T, \; \boldsymbol{\mathcal{K}} \doteq [\boldsymbol{\mathcal{K}}(t_i, t_j)]\big|_{ij, 0 \leq i, j \leq N}.$$

We learn this prior from demonstrations. Furthermore, we employ a sparse, structured GP formulation [87] as detailed in Section 5.2.

Likelihood – Information regarding a given skill generalization scenario is encoded by a collection of likelihood functions. A generalization scenario is composed of a set of random events $\mathbf{e}$, including new robot initial states, via-points, or novel obstacles in the environment. We define a likelihood function as a conditional distribution $\boldsymbol{l}(\boldsymbol{\theta}; \mathbf{e}) \propto p(\mathbf{e}|\boldsymbol{\theta})$, which assigns a probability to the occurrence of $\mathbf{e}$, given the trajectory $\boldsymbol{\theta}$,

$$p(\mathbf{e}|\boldsymbol{\theta}) \propto \exp\{-\frac{1}{2}\|\boldsymbol{h}(\boldsymbol{\theta}; \mathbf{e})\|_{\boldsymbol{\Sigma}}^2\}, \tag{5.2}$$

where $\boldsymbol{h}(\boldsymbol{\theta}; \mathbf{e})$ is a vector-valued cost function with covariance matrix $\boldsymbol{\Sigma}$. The likelihood defines feasibility for a given trajectory during skill reproduction.

MAP Inference – The optimal and feasible trajectory which generalizes the learned skill, is given by the mode of the posterior trajectory distribution. The posterior is found by conditioning the trajectory prior on the event likelihoods,

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \left\{p(\boldsymbol{\theta}|\mathbf{e})\right\} = \arg\max_{\boldsymbol{\theta}} \left\{p(\boldsymbol{\theta})p(\mathbf{e}|\boldsymbol{\theta})\right\} \tag{5.3}$$

5.2 From Stochastic Dynamical Systems To Trajectory Priors

We view robot trajectories as governed by a stochastic dynamical system. Rolling out the dynamical system results in a probability distribution on trajectories, which we treat

as a prior in our probabilistic inference framework. For continuous-time trajectories, the trajectory prior is equivalent to a Gaussian process (GP).

5.2.1 Structured Gaussian Process with Sparse Precision Matrix

In this work, we parameterize the dynamical system as a linear time-varying stochastic differential equations (LTV-SDE). The LTV-SDE results in a special class of structured GP with a sparse precision matrix (i.e. inverse covariance matrix). This sparsity can be exploited in both learning and inference for efficient computation. The LTV-SDE is defined as,

$$\dot{\boldsymbol{\theta}}(t) = \mathbf{A}(t)\boldsymbol{\theta}(t) + \mathbf{u}(t) + \mathbf{F}(t)\mathbf{w}(t), \quad \mathbf{w}(t) \sim \mathcal{GP}(\mathbf{0}, \mathbf{Q}_C(t)\delta(t-t')), \tag{5.4}$$

where $\boldsymbol{\theta}(t)$ is the instantaneous robot state consisting of vectorized current positions and their higher-order time derivatives (for all degrees of freedom), $\mathbf{u}(t)$ is a bias term, $\mathbf{A}(t)$ and $\mathbf{F}(t)$ are time-varying system matrices and $\mathbf{w}(t)$ is a white noise process with covariance $\mathbf{Q}_C(t)$. Note that a similar dynamical system has been employed in simultaneous estimation and mapping [87], and planning [84] problems. However, unlike previous approaches, the noise covariance $\mathbf{Q}_C(t)$ in our formulation, is time-varying. This provides us additional flexibility in encoding stochasticity in motions.

Taking the first and second moments of the solution to the LTV-SDE yields,

$$\boldsymbol{\mu}(t) = \boldsymbol{\Phi}(t, t_0)\boldsymbol{\mu}_0 + \int_{t_0}^{t} \boldsymbol{\Phi}(t, s)\mathbf{u}(s)ds, \tag{5.5}$$

$$\boldsymbol{\mathcal{K}}(t, t') = \boldsymbol{\Phi}(t, t_0)\mathbf{Q}_0\boldsymbol{\Phi}(t', t_0)^T + \int_{t_0}^{\min(t,t')} \boldsymbol{\Phi}(t, s)\mathbf{F}(s)\mathbf{Q}_C(s)\mathbf{F}(s)^T\boldsymbol{\Phi}(t', s)^T ds \tag{5.6}$$

where $\boldsymbol{\Phi}(t, s)$ is the state transition matrix, and $\boldsymbol{\mu}_0$ and $\mathbf{Q}_0$ are the initial mean and covariance. We may sample the GP at a finite set of times, resulting in a set of support states vectorized as $\boldsymbol{\theta} = [\boldsymbol{\theta}_0, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_N]^T$. As a direct consequence of Eq. (5.5)-(5.6), the mean, covariance and precision of $\boldsymbol{\theta}$ can be factored as,

$$\boldsymbol{\mu} = \mathbf{A}\mathbf{u}, \quad \boldsymbol{\mathcal{K}} = \mathbf{A}\mathbf{Q}\mathbf{A}^T, \quad \boldsymbol{\mathcal{K}}^{-1} = \mathbf{A}^{-T}\mathbf{Q}^{-1}\mathbf{A}^{-1}, \tag{5.7}$$

where,

$$\boldsymbol{\mu} = [\boldsymbol{\mu}(t_0), \boldsymbol{\mu}(t_1), \dots \boldsymbol{\mu}(t_N)]^T, \quad \mathbf{u} = [\boldsymbol{\mu}_0, \mathbf{u}_{0,1}, \dots \mathbf{u}_{N-1,N}]^T, \quad \mathbf{u}_{i,i+1} = \int_{t_i}^{t_{i+1}} \boldsymbol{\Phi}(t_{i+1}, s)\mathbf{u}(s)ds,$$

$$\mathbf{Q} = \text{diag}(\mathbf{Q}_0, \mathbf{Q}_{0,1}, \dots, \mathbf{Q}_{N-1,N}), \qquad \mathbf{Q}_{i,i+1} = \int_{t_i}^{t_{i+1}} \boldsymbol{\Phi}(t_{i+1}, s)\mathbf{F}(s)\mathbf{Q}_C(s)\mathbf{F}(s)^T\boldsymbol{\Phi}(t_{i+1}, s)^T ds,$$

$$\mathbf{A} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \boldsymbol{\Phi}(t_1, t_0) & \mathbf{1} & \dots & \mathbf{0} & \mathbf{0} \\ \boldsymbol{\Phi}(t_2, t_0) & \boldsymbol{\Phi}(t_2, t_1) & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \mathbf{0} & \mathbf{0} \\ \boldsymbol{\Phi}(t_{N-1}, t_0) & \boldsymbol{\Phi}(t_{N-1}, t_1) & \dots & \mathbf{1} & \mathbf{0} \\ \boldsymbol{\Phi}(t_N, t_0) & \boldsymbol{\Phi}(t_N, t_1) & \dots & \boldsymbol{\Phi}(t_N, t_{N-1}) & \mathbf{1} \end{bmatrix}.$$

For more details on the aforementioned factorization, the reader is referred to [87].

Since the matrix $\mathbf{A}$ is lower-triangular, and the matrix $\mathbf{Q}$ is block-diagonal, the resulting precision matrix $\boldsymbol{\mathcal{K}}^{-1}$ is block-tridiagonal. In Section 5.3, we exploit the sparsity of precision matrix for fast and efficient inference.

Note that in the remainder of this chapter, $\boldsymbol{\theta}(t)$ will specifically refer to a configuration space trajectory of the robot, while $\boldsymbol{x}(t)$ will refer to the robot's workspace trajectory.

5.2.2 A Combined Prior

Robot skills often impose motion constraints in the workspace[1] of the robot. To achieve this, one may choose to learn a trajectory prior in the workspace only. However, for redundant manipulators, a given workspace trajectory can result in multiple configuration space trajectories. Thus, the corresponding motion generation problem in the configuration space may be under-constrained. For redundancy resolution, it in necessary to enforce additional constraints in the configuration space of the robot. In line of this, we formulate a combined trajectory prior,

$$p_{\boldsymbol{x}}(\boldsymbol{\theta}) = p(\boldsymbol{\theta}|\boldsymbol{x}) \propto p(\boldsymbol{\theta})p(\boldsymbol{x}|\boldsymbol{\theta}). \tag{5.8}$$

The aforementioned combined trajectory prior is a distribution in configuration space, made up

[1] Our workspace formulation also applies to task space. This is without loss of generality since any task space constraint can be projected on to the workspace.

of a configuration space prior $p(\boldsymbol{\theta})$ conditioned on a workspace prior $p(\boldsymbol{x}|\boldsymbol{\theta})$. One may choose to learn either or both components of the combined prior. In practice, we learn the workspace prior from demonstrations and hand-specify the configuration space prior to ensure smooth motions. Perhaps the most straightforward choice of configuration space prior is the constant-velocity prior [87, 84], which forces minimum accelerations in the configuration space.

5.2.3 Learning Workspace Prior from Demonstrations

The workspace prior distribution in (5.8) is defined as

$$p(\boldsymbol{x}|\boldsymbol{\theta}) \propto \exp\{-\frac{1}{2}||\boldsymbol{\psi}(\boldsymbol{\theta}) - \boldsymbol{\mu}^{\boldsymbol{x}}||^2_{\boldsymbol{\mathcal{K}}^{\boldsymbol{x}}}\}, \tag{5.9}$$

where the function ψ maps a trajectory in configuration space to a workspace trajectory, and the hyper-parameters $\boldsymbol{\mu}^{\boldsymbol{x}}$ and $\boldsymbol{\mathcal{K}}^{\boldsymbol{x}}$ are the mean and the covariance of the workspace distribution. Our goal is to learn these hyper-parameters from human demonstrations in workspace.

In practice, we only have access to the workspace support states $\boldsymbol{x}_i$ at discrete time instances. Thus, we employed a discrete version of the LTV-SDE in (5.4) for the experiments we considered,

$$\boldsymbol{x}_{i+1} = \boldsymbol{\Phi}^{\boldsymbol{x}}(t_{i+1}, t_i)\boldsymbol{x}_i + \mathbf{u}^{\boldsymbol{x}}_{i,i+1} + \mathbf{w}^{\boldsymbol{x}}_{i,i+1}, \quad \mathbf{w}^{\boldsymbol{x}}_{i,i+1} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}^{\boldsymbol{x}}_{i,i+1}), \tag{5.10}$$

where the unknown parameters $\boldsymbol{\Phi}^{\boldsymbol{x}}(t_{i+1}, t_i)$, $\mathbf{u}^{\boldsymbol{x}}_{i,i+1}$ and $\mathbf{Q}^{\boldsymbol{x}}_{i,i+1}$ are the transition matrix, the bias term, and the noise covariance matrix in workspace. Note that the aforementioned parameters fully define the workspace trajectory prior as given by (5.7). Therefore, the problem of learning the trajectory prior is equivalent to learning the LTV-SDE parameters.

Given M trajectory demonstrations $\mathbf{X} = \{\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^M\}$, the regularized maximum likelihood estimate of the unknown parameters for the time interval $[t_i, t_{i+1}]$ is given by,

$$\boldsymbol{\Phi}^{\boldsymbol{x}}(t_{i+1}, t_i),\ \mathbf{u}^{\boldsymbol{x}}_{i,i+1} = \underset{\mathbf{u}^{\boldsymbol{x}}_{i,i+1},\ \boldsymbol{\Phi}^{\boldsymbol{x}}(t_{i+1},t_i)}{\arg\min} \sum_{m=1}^{M} \|\mathbf{r}^m_{i,i+1}\|^2 + \lambda\|\boldsymbol{\Phi}^{\boldsymbol{x}}(t_{i+1}, t_i)\|^2_F, \tag{5.11}$$

$$\mathbf{Q}^{\boldsymbol{x}}_{i,i+1} = \frac{1}{M}\sum_{m=1}^{M} \mathbf{r}^m_{i,i+1}\mathbf{r}^{m^T}_{i,i+1}, \tag{5.12}$$

where the residual $\mathbf{r}^m_{i,i+1} = \mathbf{u}^{\boldsymbol{x}}_{i,i+1} - \boldsymbol{x}^m_{i+1} + \boldsymbol{\Phi}^{\boldsymbol{x}}(t_{i+1}, t_i)\boldsymbol{x}^m_i$ and λ is the regularization parameter. The hyper-parameters of the prior are calculated using the relationships in (5.7). The computational complexity associated with learning the workspace prior is $O(N \cdot d^3)$, where N is the number of support states comprising the trajectory and d is the dimensionality of each support state.

5.3 Efficient Inference via Factor Graphs

In this section, we show how to exploit the sparsity of the underlying system to efficiently carry out MAP inference using the learned prior, to reproduce the skill. Any probability distribution P can be represented as a product of functions organized as a bipartite *factor graph* [88] $G = \{\boldsymbol{\Theta}, \mathcal{F}, \mathcal{E}\}$,

$$P(\boldsymbol{\Theta}) \propto \prod_i f_i(\boldsymbol{\Theta}_i), \tag{5.13}$$

where a set of random variables $\boldsymbol{\Theta} \doteq \{\boldsymbol{\theta}_i\}$ and a set of *factors* $\mathcal{F} \doteq \{f_i\}$ which are functions on variable subsets $\boldsymbol{\Theta}_i$, are connected by a set of edges $\mathcal{E}$. The structure of the precision matrix of a distribution is captured by the structure of its factor graph, i.e. a sparser precision matrix leads to a more factorized distribution. Efficiency during inference is a direct result of this factorization. In the remainder of this section we will present the factor graph formulation of our problem, the factors used in our implementation and how the inference is performed via efficient nonlinear least squares optimization.

5.3.1 Prior Factors

The combined prior in (5.8) can be factored as

$$p_{\boldsymbol{x}}(\boldsymbol{\theta}) \propto p(\boldsymbol{\theta})p(\boldsymbol{x}|\boldsymbol{\theta}) \propto f^{\text{gp},\boldsymbol{\theta}} f^{\text{gp},\boldsymbol{x}} = \prod_{i=0}^{N-1} f_i^{\text{gp},\boldsymbol{\theta}}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{i+1})\ f_i^{\text{gp},\boldsymbol{x}}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{i+1}), \tag{5.14}$$

where,

$$f_i^{\text{gp},\boldsymbol{x}}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{i+1}) = \exp\left\{-\frac{1}{2}\|\boldsymbol{\Phi}^{\boldsymbol{x}}(t_{i+1}, t_i)\boldsymbol{\psi}(\boldsymbol{\theta}_i) - \boldsymbol{\psi}(\boldsymbol{\theta}_{i+1}) + \mathbf{u}_{i,i+1}^{\boldsymbol{x}}\|^2_{\mathbf{Q}_{i,i+1}^{\boldsymbol{x}}}\right\}$$

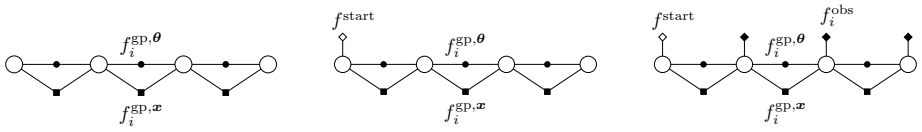
are the workspace prior factors learned from demonstrations as described in Section 5.2.3, and

$$f_i^{\text{gp},\boldsymbol{\theta}}(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{i+1}) = \exp\left\{-\frac{1}{2}\|\boldsymbol{\Phi}^{\boldsymbol{\theta}}(t_{i+1}, t_i)\boldsymbol{\theta}_i - \boldsymbol{\theta}_{i+1} + \mathbf{u}_{i,i+1}^{\boldsymbol{\theta}}\|^2_{\mathbf{Q}_{i,i+1}^{\boldsymbol{\theta}}}\right\}, \tag{5.15}$$

are the pre-specified smoothness prior factors in configuration space (see Section 5.2.2).

5.3.2 Likelihood Factors

The factorization of the likelihood is problem-specific and depends on the events being considered. In this work. we only consider events involving different start conditions and/or



(a) (b) (c)

Figure 5.2: Example factor graphs. States $\boldsymbol{\theta}_i$ are shown as white circles.

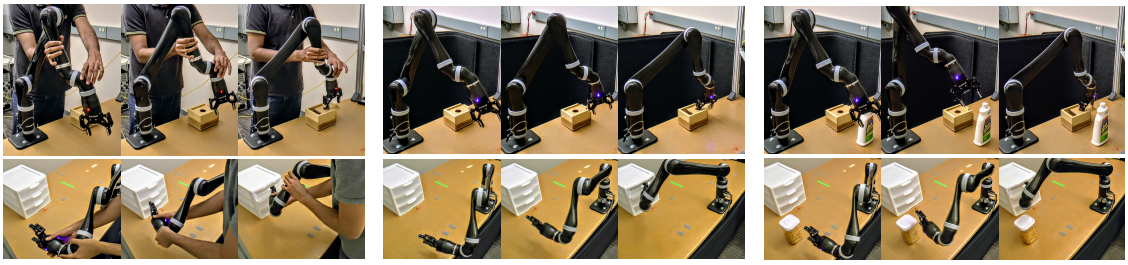
collision avoidance. Figure 5.2(b) shows the joint distribution of the prior and *start-state* likelihood. The posterior inference involves conditioning the prior on a desired start state,

$$p(\mathbf{e}|\boldsymbol{\theta}) \propto f^{\text{start}} = \exp\left\{-\frac{1}{2}\|\boldsymbol{\theta}_0 - \boldsymbol{\theta}_{start}\|^2_{\boldsymbol{\sigma}_{start}}\right\}, \tag{5.16}$$

where a very small covariance $\boldsymbol{\sigma}_{start}$ signifies the certainty of finding a solution that starts from a desired start state $\boldsymbol{\theta}_{start}$. Figure 5.2(c) shows the joint distribution with an additional *collision-free* likelihood. The posterior and associated likelihood are then defined as,

$$p(\mathbf{e}|\boldsymbol{\theta}) \propto f^{\text{start}} f^{\text{obs}} = f^{\text{start}}(\boldsymbol{\theta}_0) \prod_{i=1}^{N} f_i^{\text{obs}}(\boldsymbol{\theta}_i), \quad f_i^{\text{obs}}(\boldsymbol{\theta}_i) = \exp\left\{-\frac{1}{2} \parallel \mathbf{h}(\boldsymbol{\theta}_i) \parallel^2_{\boldsymbol{\sigma}_{obs}}\right\}, \tag{5.17}$$

where f_i^{obs} are unary obstacle factors. The collision for any state is evaluated with a precomputed signed distance field, a cost function $\mathbf{h}$, and a hyperparameter $\boldsymbol{\sigma}_{obs}$ that balances the weight on collision avoidance versus staying close to the prior. This technique is also used in GPMP2 for collision avoidance during motion planning (see [89] for details). It is worth noting that, due to this generic formulation, the learned skills can be reproduced in any new environment with never-before-seen obstacles as long as a signed distance field is calculated beforehand.



(a) Kinesthetic demonstration (b) Reproduction (c) Reproduction with obstacle

Figure 5.3: Demonstration and reproduction of *box-opening*(top) and *drawer-opening*(bottom)

5.3.3 Efficient Inference

Finally, for efficient MAP inference, we take the negative log of the posterior distribution $p(\boldsymbol{\theta}|\mathrm{e}) \propto p_{\boldsymbol{x}}(\boldsymbol{\theta})p(\mathrm{e}|\boldsymbol{\theta})$ using the combined prior (5.8),

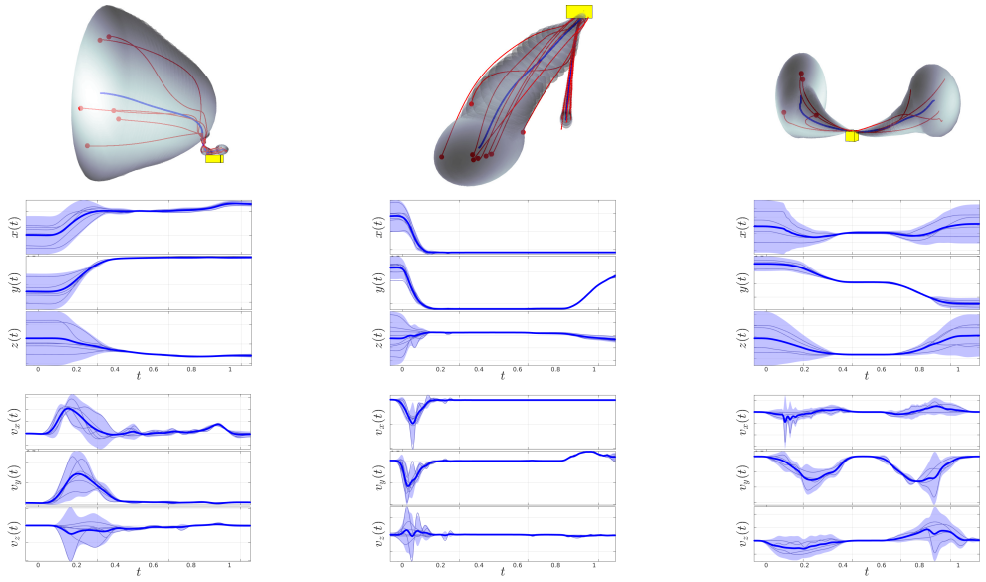
$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \left\{ \frac{1}{2} \parallel \boldsymbol{\theta} - \boldsymbol{\mu}^{\boldsymbol{\theta}} \parallel^2_{\boldsymbol{\mathcal{K}}^{\boldsymbol{\theta}}} + \frac{1}{2} \parallel \boldsymbol{\psi}(\boldsymbol{\theta}) - \boldsymbol{\mu}^{\boldsymbol{x}} \parallel^2_{\boldsymbol{\mathcal{K}}^{\boldsymbol{x}}} + \frac{1}{2} \parallel \boldsymbol{h}(\boldsymbol{\theta}; \boldsymbol{e}) \parallel^2_{\boldsymbol{\Sigma}}, \right\} \tag{5.18}$$

Thus, giving a nonlinear least squares optimization based formulation for the inference problem. The factor graph allows us to compactly organize the computation, with optimization performed using Gauss-Newton or Levenberg-Marquardt. Combining the structure exploiting inference and the quadratic convergence rates of the optimization, make this approach computationally efficient. The computational complexity is directly related to how well the distributions factorize, and since only unary or binary factors are present, the problem is extremely sparse and thus very efficient to solve. The complexity is $O(N \cdot d^3)$, where N is the number of states and d is the dimensionality of the state. Although the complexity would increase in the presence of further higher-order factors that define costs across multiple states.

5.4 Experimental Results

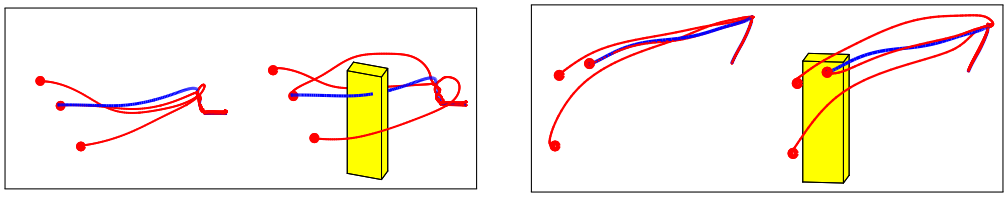
We implemented CLAMP in C++ and MATLAB[2], and tested it on multiple manipulation problems executed by a Kinova JACO[2] 6-DOF arm. For our experiments, we defined instantaneous workspace state $\boldsymbol{x}(t) \in \mathbb{R}^6$ as a vector concatenation of end-effector position

[2]Code: https://github.com/gt-rail/clamp.



(a) *box-opening* skill (b) *drawer-opening* skill (c) *picking* skill

Figure 5.4: Workspace priors. The mean is in blue with an envelope showing the 95% confidence.



(a) *box-opening* skill (b) *drawer-opening* skill

Figure 5.5: Motions in red from different initial states. The obstacle is in yellow and the prior position mean is in blue.

and velocity. Furthermore, the configuration space state $\boldsymbol{\theta}(t) \in \mathbb{R}^{12}$ was made up of joint positions and velocities. Similar to [84], we employed the constant velocity prior for $f^{\text{gp},\boldsymbol{\theta}}$, encouraging smoothness in configuration space. The accompanying video[3] shows the experimental results.

We validated our proposed method on three skills including, *box-opening*, *drawer-opening* and *picking*. For each skill, we provided multiple demonstrations with different initial end-effector states (varying initial position, zero initial velocity) through kinesthetic teaching [1]. The end-effector positions over time were recorded and the trajectories were

[3]Video of experiments: https://youtu.be/DDs_ZxsN0Ek

temporally aligned using dynamic time warping [90]. The corresponding end-effector linear velocities were estimated by fitting a cubic spline and differentiating with respect to time. Figure 5.4 shows the learned prior distributions i.e. the skill models.

For the *box-opening* skill, each demonstration was composed of two primitive actions, reaching and sliding the lid of the box. The sliding part of the skill was more constrained compared to the reaching part. As shown in Figure 5.4(a), the variance in the state variables (i.e. positions and velocities) become much smaller during the sliding portion of the trajectory. For the *drawer-opening* skill, each demonstration involves reaching the drawer handle and pulling it in the direction perpendicular to the drawer body. Like the *box-opening* skill, the second part of the demonstrations were highly restrictive in both positions and velocities to satisfy skill completion, as shown in Figure 5.4(b). Finally, the *picking* skill involved reaching an object from different initial end-effector positions and then placing it at different locations. As shown in Figure 5.4(c), since object location was fixed across all demonstrations, the variance in the position state variable is much smaller in the middle part of the skill. However, compared to the other two skills which deal with articulated object manipulation, the velocity profile is not as critical for the *picking* skill. For all the skills, the prior also encodes the coupling between the state variables. This is a consequence of the underlying LTV-SDE.

Provided the initial state of the robot, the likelihood in (5.16) was used during inference to find MAP trajectories for skill reproduction. For obstacle avoidance, we further incorporated the likelihood in (5.17). $\boldsymbol{\sigma}_{obs}$ was set manually to enable the desired clearance of the robot from the obstacle. In general, $\boldsymbol{\sigma}_{obs}$ depends on the size of the robot, desired clearance and the environment itself. The MAP trajectories for all scenarios were found using factor graph optimization to solve (5.18).

5.5 Conclusion

We have presented CLAMP, a novel approach which unifies LfD and inference-based planning. Within this approach, we first learn the skill as a stochastic dynamical system. Next, we carry out fast numerical optimization over factor graphs for efficient inference. Using this approach, we managed to generate trajectories that are optimal with respect to the learned skill (i.e. the trajectory prior) and feasible with respect to the reproduction scenario composed of various events (i.e. the likelihood). Although in our current implementation, we consider robot trajectories to be comprised of positions and velocities and the events to be made up of robot's current initial state and obstacle clearance, our approach allows incorporation of further higher-order dynamics or event likelihoods. We have provided experimental validation of our approach in learning and generalizing object manipulation skills, even in the presence of new obstacles.

CHAPTER 6

LEARNING TRAJECTORY PRIORS FROM DEMONSTRATIONS IN CLUTTER

Most prior LfD approaches [11, 8, 9, 12] are based on the assumption that demonstrations can be performed in uncluttered, minimally constrained environments. The presence of clutter in the demonstration environments can introduce additional constraints on human demonstrations that are unrelated to the target skill or the underlying human intent. If unaccounted for, this can lead to suboptimal skill models. However, restructuring the world to remove clutter is often impractical, which limits the viability of such approaches.

In this work, we tackle the problem of learning skills from a set of demonstrations, which can be partially or fully influenced by the presence of obstacles (see Fig. 6.1). To contend with obstacles during training, we present *importance weighted skill learning*. Specifically, we adopt and extend the CLAMP framework, proposed in Chapter 5, to utilize demonstrations from cluttered environments.

Importance weighted skill learning (see Fig. 6.2) rates the importance of demonstration trajectories while learning the trajectory prior. We propose an importance weighting function that assigns lower importance to parts of demonstrations that are more likely to be influenced by obstacles. We present batch and incremental versions of our algorithm: batch learning is

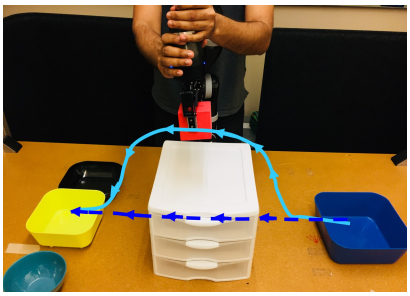


Figure 6.1: A human is demonstrating a placing skill, which involves placing the (red) cube from the (blue) bowl on the right in to one of the three bowls on the left. The figure contrasts the demonstrated trajectory (light blue), which is influenced by an obstacle (drawer) in the environment, with the intended straight-line trajectory (dark blue) in the absence of the obstacle.

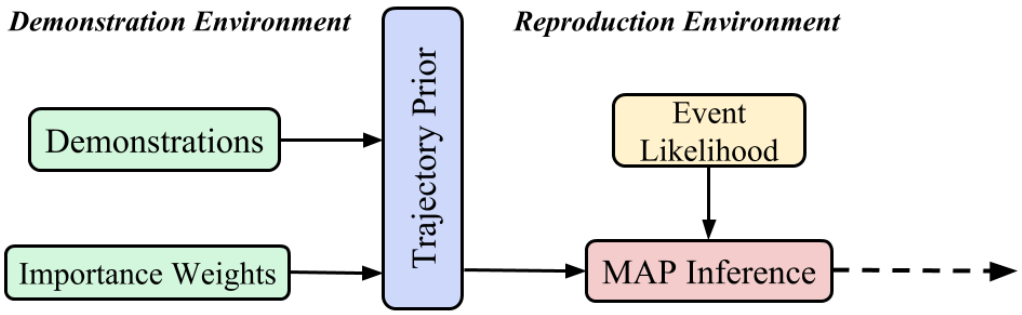


Figure 6.2: An overview of importance weight skill learning.

useful when the set of initial demonstrations are sufficient for learning a reasonable skill model, while incremental learning is useful in scenarios that require refinement of the skill model as new demonstrations in new environments become available.

We validate our approach on a 7-DOF JACO^2 manipulator with *reaching* and *placing* skills. In all the experiments, we evaluate the approach by providing demonstrations in cluttered environments and then changing the environments for reproduction.

6.1 Related Work

Many existing approaches to trajectory-based LfD address the problem of avoiding obstacles in the reproduction scenario. Some approaches add obstacle avoidance in the skill reproduction phase as a reactive strategy [29, 26, 91], while others carry out motion planning or trajectory optimization [14, 46, 13, 80]. In all these approaches, the skill model is learned from demonstrations that are not affected by obstacles. Any constraints or costs associated with obstacles are typically present during reproduction only. However, in an obstacle-rich environment, the demonstrations themselves are likely to be influenced by the presence of obstacles, which could have repercussions during skill reproduction.

There have been a few attempts to address the problem of learning skills from demonstrations in cluttered environments. For example, [92, 93] learn a dynamic movement primitive (DMP) as well as a coupling term for obstacle avoidance from demonstrations. These approaches suffer from two major problems. First, since DMPs follow a single

demonstration, they fail to learn potentially different ways of executing the skill, thereby limiting its robustness in new scenarios. Second, due to the reactive nature of the obstacle avoidance strategy, the reproduced trajectory does not necessarily preserve the shape of the motion in the presence of obstacles. Ghalamzan et al. [47], proposed an approach based on learning a cost functional from human demonstrations. This cost functional is dependent on two components: the deviation from the mean of the demonstrations, and the distance from obstacles in the environment. Parameters of both these components are estimated from human demonstrations. A major drawback of this approach is the assumption that the mean of the demonstrations sufficiently expresses the demonstrated skill. This assumption however stands invalid for skills which can be executed in multiple ways and hence requires a more expressive skill model.

Our proposed method is based on learning an underlying stochastic dynamical system from demonstrations. Depending on the part of the state-space the robot lies in, this dynamical system is able to generate different ways of executing a learned skill. We make use of importance weighting to discount the effect of obstacles that are present when the demonstrations are provided. Specifically, the parts of demonstrations in the vicinity of obstacles are penalized to account for their deviation from the desired skill or the human intention.

6.2 Importance Weighted Trajectory Prior Learning

In this section, we introduce importance weighting learning of the trajectory prior in CLAMP. The importance weighting enables excluding the effects of unwanted influences on demonstrations. Similar to CLAMP, we estimate the parameters of the skill dynamics model in Eq. (5.10) from demonstrations. As a preliminary step, lets re-write (5.10),

$$\boldsymbol{x}_{i+1} = \tilde{\boldsymbol{\Phi}}_{i+1}\tilde{\boldsymbol{x}}_i + \mathbf{w}_{i+1}, \quad \mathbf{w}_{i+1} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_{i+1}) \tag{6.1}$$

where, $\tilde{\boldsymbol{x}}_i = [\mathbf{1}, \boldsymbol{x}_i]^T$, and $\tilde{\boldsymbol{\Phi}}_{i+1} = [\mathbf{u}_{i+1}, \boldsymbol{\Phi}_{i+1}]$. Additionally, we define an importance weighting function as $w : \mathbb{R}^d \rightarrow \mathbb{R}$. The importance weighting function should give higher weights to robot states that are less likely to deviate from the skill constraints or the true human intent. While this importance weighting formulation can be used in other contexts too, in this chapter we define a specific form of importance weighting to account for the influence of unwanted obstacles in the demonstration environment. The exact form of this environment-dependent obstacle weighting function is presented in Section 6.3.

6.2.1 Batch Learning

Let's assume the availability of K trajectory demonstrations, with the k^{th} demonstration defined as $\boldsymbol{x}^k = [\boldsymbol{x}_0^k, \boldsymbol{x}_1^k, \ldots, \boldsymbol{x}_N^k]^T$. For each discrete time interval $(t_i, t_{i+1}]$, the inputs are collected into a matrix $\tilde{\boldsymbol{X}}_i = [\tilde{\boldsymbol{x}}_i^1, \tilde{\boldsymbol{x}}_i^2, \ldots, \tilde{\boldsymbol{x}}_i^K]$ while the corresponding targets into a matrix $\boldsymbol{X}_{i+1} = [\boldsymbol{x}_{i+1}^1, \boldsymbol{x}_{i+1}^2, \ldots, \boldsymbol{x}_{i+1}^K]$. Furthermore, the matrix $\mathbf{W}_i = \text{diag}\big(w(\boldsymbol{x}_i^1), w(\boldsymbol{x}_i^2), \ldots, w(\boldsymbol{x}_i^K)\big)$ defines a state-dependent importance weight matrix.

The batch skill learning formulation seeks to find $\tilde{\boldsymbol{\Phi}}_{i+1}$ and $\mathbf{Q}_{i+1}$, which minimize a regularized squared norm over the provided demonstrations.

$$
\begin{aligned}
&\tilde{\boldsymbol{\Phi}}_{i+1}^*, \mathbf{Q}_{i+1}^* \\
&\quad = \underset{\tilde{\boldsymbol{\Phi}}_{i+1}, \mathbf{Q}_{i+1}}{\arg\min} \left\{ \mathcal{L}(\tilde{\boldsymbol{\Phi}}_{i+1}, \mathbf{Q}_{i+1}) \right\} \\
&\quad = \underset{\tilde{\boldsymbol{\Phi}}_{i+1}, \mathbf{Q}_{i+1}}{\arg\min} \left\{ \text{tr}\big(\mathbf{Q}_{i+1}^{-1} \mathbf{E}_{i+1} \mathbf{W}_i \mathbf{E}_{i+1}^T\big) + \lambda \|\tilde{\boldsymbol{\Phi}}_{i+1}\|_F^2 \right\}
\end{aligned}
\tag{6.2}
$$

where $\mathbf{E}_{i+1} = \boldsymbol{X}_{i+1} - \tilde{\boldsymbol{\Phi}}_{i+1} \tilde{\boldsymbol{X}}_i$ defines the error matrix, and λ is a regularization coefficient.

The solution to the batch skill learning problem in (6.2) is given by the weighted ridge

regression estimate,

$$\tilde{\mathbf{\Phi}}^*_{i+1} = \boldsymbol{X}^T_{i+1}\mathbf{W}_i\tilde{\boldsymbol{X}}_i\big(\tilde{\boldsymbol{X}}_i\mathbf{W}_i\tilde{\boldsymbol{X}}_i^T + \lambda\mathbf{I}\big)^{-1}, \tag{6.3}$$

$$\mathbf{Q}^*_{i+1} = \frac{1}{z}\mathbf{E}^*_{i+1}\mathbf{W}_i{\mathbf{E}^*_{i+1}}^T, \tag{6.4}$$

$$z = \frac{\text{tr}(\mathbf{W}_i)^2 - \text{tr}(\mathbf{W}_i^T\mathbf{W}_i)}{\text{tr}(\mathbf{W}_i)}.$$

6.2.2 Incremental Learning

The batch skill learning procedure assumes that there are enough demonstrations available to learn an optimal skill model. However, as more demonstrations are aggregated over time, possibly in different environments, it is desirable to refine the model since more data provides a better estimate of the skill. To achieve this, we propose incremental weighted skill learning.

Our incremental skill learning procedure is based on Bayesian inference. In this formulation, we maintain a joint probability distribution over the unknown skill dynamics parameters. Every time a new demonstration is collected, a posterior over the skill dynamics parameters is calculated

$$\begin{aligned} &p(\tilde{\mathbf{\Phi}}_{i+1}, \mathbf{Q}_{i+1}|\mathcal{D}^{1:k}) \\ &\quad = p(\mathcal{D}^k|\tilde{\mathbf{\Phi}}_{i+1}, \mathbf{Q}_{i+1})p(\tilde{\mathbf{\Phi}}_{i+1}, \mathbf{Q}_{i+1}|\mathcal{D}^{1:k-1}), \end{aligned} \tag{6.5}$$

where $\mathcal{D}^{1:k} = \{\{\tilde{\boldsymbol{x}}_i^1, \boldsymbol{x}_{i+1}^1\}, \{\tilde{\boldsymbol{x}}_i^2, \boldsymbol{x}_{i+1}^2\}, \ldots, \{\tilde{\boldsymbol{x}}_i^k, \boldsymbol{x}_{i+1}^k\}\}$. At any stage, the mode of the posterior distribution provides an estimate of the unknown parameters.

Skill Dynamics Distribution

The joint probability distribution over the unknown parameters $\tilde{\mathbf{\Phi}}_{i+1}$ and $\mathbf{Q}_{i+1}$ is given by

$$p(\tilde{\mathbf{\Phi}}_{i+1}, \mathbf{Q}_{i+1}) = p(\tilde{\mathbf{\Phi}}_{i+1}|\mathbf{Q}_{i+1})p(\mathbf{Q}_{i+1}), \tag{6.6}$$

where,

$$p(\tilde{\mathbf{\Phi}}_{i+1}|\mathbf{Q}_{i+1}) = \mathcal{MN}(\boldsymbol{M}_{i+1}, \mathbf{Q}_{i+1}, \boldsymbol{R}_{i+1}), \tag{6.7}$$

$$p(\mathbf{Q}_{i+1}) = \mathcal{W}^{-1}(\boldsymbol{V}_{i+1}, \nu_{i+1}), \tag{6.8}$$

$\mathcal{MN}$ refers to a matrix-normal distribution with matrix-valued mean $\boldsymbol{M}_{i+1}$ and covariances $\mathbf{Q}_{i+1}$ and $\boldsymbol{R}_{i+1}$ for the rows and columns respectively. $\mathcal{W}^{-1}$ refers to an inverse-Wishart distribution with positive definite scale matrix $\boldsymbol{V}_{i+1}$ and ν_{i+1} degrees of freedom. Note that matrix-normal and inverse-Wishart distributions are generalizations of the normal and inverse-gamma distributions respectively to the multivariate case.

Demonstration Likelihood

The likelihood of observing the input-target pair from the k^{th} demonstration under the stochastic dynamics (6.1) is given by

$$\begin{aligned} p(\mathcal{D}^k|\tilde{\mathbf{\Phi}}_{i+1}, \mathbf{Q}_{i+1}) &\doteq p(\boldsymbol{x}_{i+1}^k|\tilde{\boldsymbol{x}}_i^k, \tilde{\mathbf{\Phi}}_{i+1}, \mathbf{Q}_{i+1}) \\ &\propto \exp\left\{ -\frac{1}{2}(w_i^k \mathbf{Q}_{i+1}^{-1} \mathbf{e}_{i+1}^k \mathbf{e}_{i+1}^{k\ T}) \right\}. \end{aligned} \tag{6.9}$$

where $\mathbf{e}_{i+1}^k = \boldsymbol{x}_{i+1}^k - \tilde{\mathbf{\Phi}}_{i+1}\tilde{\boldsymbol{x}}_i^k$ and $w_i^k = w(\boldsymbol{x}_i^k)$. Note that the likelihood is scaled by the weight in order to incorporate the importance weighting.

Skill Dynamics Inference

The skill dynamics parameters after assimilation of k demonstrations is given by the mode of the joint posterior distribution (*maximum a posteriori*),

$$\tilde{\mathbf{\Phi}}_{i+1}^k, \mathbf{Q}_{i+1}^k = \underset{\tilde{\mathbf{\Phi}}_{i+1}, \mathbf{Q}_{i+1}}{\arg\max} \left\{ p(\tilde{\mathbf{\Phi}}_{i+1}, \mathbf{Q}_{i+1}|\mathcal{D}^{1:k}) \right\}. \tag{6.10}$$

Due to the properties of matrix-normal and inverse Wishart distributions, the mode of the joint distribution turns out to be equivalent to the product of the modes of the two conditional distributions [94],

$$\tilde{\mathbf{\Phi}}_{i+1}^{k} = \arg\max_{\tilde{\mathbf{\Phi}}_{i+1}} \left\{ p(\tilde{\mathbf{\Phi}}_{i+1} | \mathbf{Q}_{i+1}, \mathcal{D}^{1:k}) \right\} = \boldsymbol{M}_{i+1}^{k} \tag{6.11}$$

$$\mathbf{Q}_{i+1}^{k} = \arg\max_{\mathbf{Q}_{i+1}} \left\{ p(\mathbf{Q}_{i+1} | \mathcal{D}^{1:k}) \right\} = \frac{1}{\nu_{i+1}^{k} + D + 1} \boldsymbol{V}_{i+1}^{k}. \tag{6.12}$$

Furthermore, the parameters of the conditional distributions are governed by the following update laws,

$$\begin{aligned}
\boldsymbol{R}_{i+1}^{k} &= w_i \tilde{\boldsymbol{x}}_i \tilde{\boldsymbol{x}}_i^T + \boldsymbol{R}_{i+1}^{k-1} \\
\boldsymbol{M}_{i+1}^{k} &= (w_i \boldsymbol{x}_{i+1} \tilde{\boldsymbol{x}}_i^T + \boldsymbol{M}_{i+1}^{k-1} \boldsymbol{R}_{i+1}^{k-1})(\boldsymbol{R}_{i+1}^{k})^{-1} \\
\boldsymbol{V}_{i+1}^{k} &= \begin{array}{l} \boldsymbol{V}_{i+1}^{k-1} + w_i (\boldsymbol{x}_{i+1} - \boldsymbol{M}_{i+1}^{k} \tilde{\boldsymbol{x}}_i)(\boldsymbol{x}_{i+1} - \boldsymbol{M}_{i+1}^{k} \tilde{\boldsymbol{x}}_i)^T \\ \qquad + (\boldsymbol{M}_{i+1}^{k} - \boldsymbol{M}_{i+1}^{k-1}) \boldsymbol{R}_{i+1}^{k-1} (\boldsymbol{M}_{i+1}^{k} - \boldsymbol{M}_{i+1}^{k-1})^T \end{array} \\
\nu_{i+1}^{k} &= 1 + \nu_{i+1}^{k-1}
\end{aligned}$$

The incremental learning procedure is initialized with a prior joint distribution $p(\tilde{\mathbf{\Phi}}_{i+1}, \mathbf{Q}_{i+1} | \phi)$. The Gaussian component of the joint prior is selected to be the ridge regression prior, that is, $\boldsymbol{M}_{i+1}^{0} = \mathbf{0}$ and $\boldsymbol{R}_{i+1}^{0} = \frac{1}{\alpha} \boldsymbol{I}$. The inverse Wishart component is selected to be an uninformed prior, with $\boldsymbol{V}_{i+1}^{0} = \frac{1}{\beta} \boldsymbol{I}$ and $\nu_{i+1}^{0} = \frac{1}{\beta}$. Here α and β are positive scalars. In our implementation, we set $\alpha = \beta = 10^{10}$. Note that smaller values of these scalars makes the prior too strict, which restrains the skill model from fitting the data well.

6.3 Environment-dependent importance weighting function

In this section, we define the importance weighting function to enable skill learning from demonstrations, which may be provided in the presence of obstacles in the environment. The weighting function gives lower importance to the parts of a demonstration which are

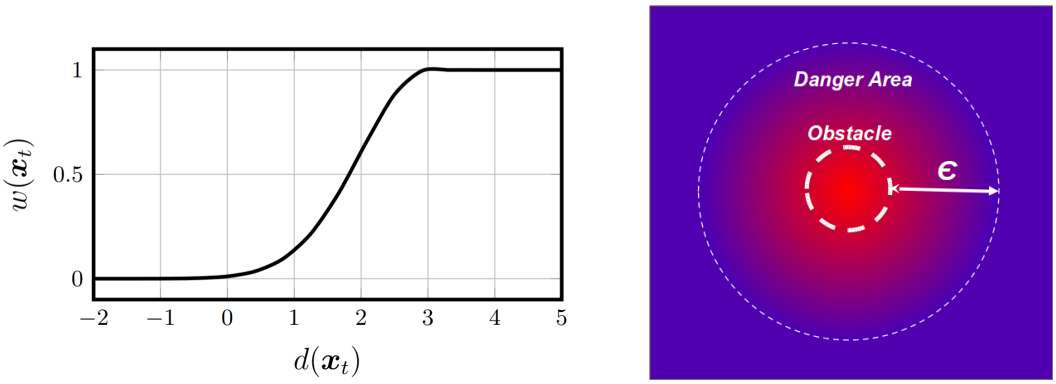


Figure 6.3: An illustration of an importance weight function parameterized by $\epsilon = 3$ and $\sigma_{obs} = 1$ (left) and a signed distance field (right). The importance weight levels at 1 outside the danger area, and decays down to zero inside with the slope governed by σ_{obs}.

more likely to be influenced by the presence of an obstacle and therefore deviate from the intent of the human.

We hypothesize that the parts of demonstrations closer to obstacles are influenced by the obstacles and therefore fail to satisfy the skill constraints. Conversely, partial trajectories farther away from obstacles are more likely to satisfy the skill constraints and should be given more importance. For a given state $\boldsymbol{x}_i$, we define the importance weight to be equivalent to the likelihood of staying collision-free [84]. For this likelihood function, we first define a hinge loss function

$$c(\boldsymbol{x}_i) = \begin{cases} -d(\boldsymbol{x}_i) + \epsilon & d(\boldsymbol{x}_i) \leq \epsilon \\ 0 & d(\boldsymbol{x}_i) > \epsilon \end{cases},$$

where $d(\cdot)$ is the signed distance from the closest obstacle in an environment and ϵ specifies the 'danger area' around the obstacle. With this hinge loss, we assume that an obstacle affects a state only when it is within the danger area around the obstacle. Outside of this danger area, the obstacle has no influence on the state. The importance weight itself is given by a function in the exponential family,

$$w(\boldsymbol{x}_i) = \exp\left\{ -\frac{c(\boldsymbol{x}_i)^2}{2\sigma_{obs}^2} \right\}, \tag{6.13}$$

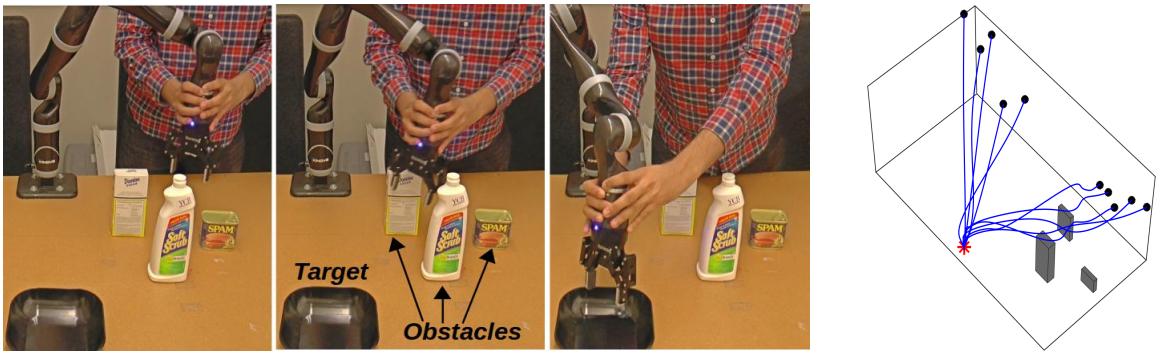


Figure 6.4: Human demonstrations for the *reaching* skill. All demonstrations reach the bowl from different initial positions in the presence of three obstacles in the environment. *Left*: Snapshots of a demonstrations avoiding the obstacles. *Right*: A 3-D plot showing all the demonstrations and the obstacles.

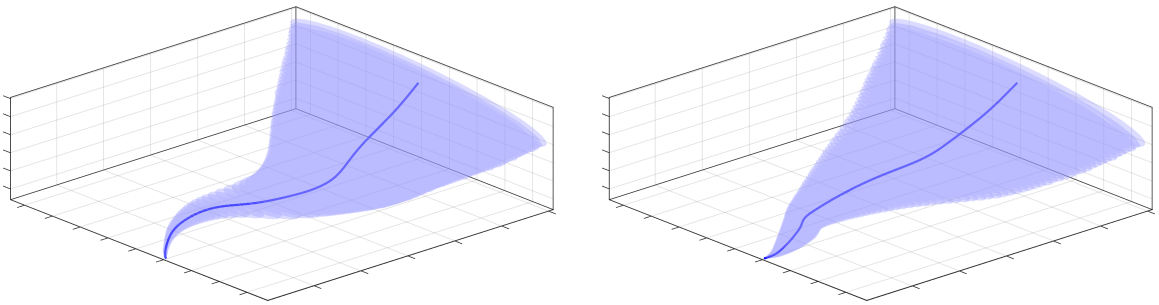
where the parameter σ_{obs} dictates the rate of decay of the importance weight for states within the 'danger area'. The smaller the value of σ_{obs}, the faster the importance weight will decay down to zero (see Fig 6.3).

6.4 Experiments

We evaluate the performance of our method on two different skills[1]: 1) the *reaching* skill, and 2) the *placing* skill. For both skills, a human provides multiple demonstrations via kinesthetic teaching on a 7-DOF JACO[2] manipulator. The end-effector positions are recorded and the corresponding instantaneous velocities are estimated by fitting a cubic spline to each demonstration and taking its time derivative. Furthermore, the demonstrations are also time-aligned using dynamic time warping (DTW). To setup the trajectory prior in (5.1), we define the robot states $\boldsymbol{x}_i$ as the vector concatenation of instantaneous robot positions and velocities.

For the *reaching* skill, the goal is to reach an object from different locations. Hence, all the demonstrations share the same goal state while the initial state varies. In the absence of any obstacles in the path, a demonstration follows a nearly straight-line path to the goal. In the presence of obstacles in the path, the demonstrations deviate from this desired path in

[1] Accompanying video: https://youtu.be/03r8Tblhq7k



(a) without importance weighting (b) with importance weighting

Figure 6.5: Trajectory prior visualization for the *reaching* skill. The blue line is the mean of the prior, and the blue shaded region shows one standard deviation around the mean.

order to avoid collision with the obstacles. Fig. 6.4 shows the demonstration environment and the corresponding demonstrations.

In order to learn the trajectory prior for this skill, we use importance weighted skill learning, as described in Section 6.2.1. The demonstrations reaching the target from the uncluttered part of the environment represent the true human intent. Therefore, we expect our trajectory prior to be biased towards these demonstrations. Fig. 6.5 shows the trajectory distributions (i.e. time-evolving state distributions) encoded in the trajectory priors learned with and without importance weighting. The trajectory distributions are generated by rolling out the stochastic skill dynamics in (6.1) with an initial state distribution given by a Gaussian over the initial demonstration states. The mean of the trajectory distribution generated with importance weighting deviates less from the intended straight-line path, exhibiting the true underlying skill, as compared to the distribution without importance weighting. To enable this, we empirically selected the parameters of the importance weight function in (6.13), such that the parts of state-space likely to be under obstacle influence can be successfully downplayed while learning the prior. A value of $\epsilon = 0.3m$ and $\sigma_{obs} = 0.01m$ provided sufficient bounding region around the obstacles in most cases.

Fig. 6.6 shows multiple instances of reproduction for the *reaching* skill. The skill is reproduced with (5.3) by conditioning the learned trajectory prior on the likelihood of starting from a desired initial state and the likelihood of staying clear of arbitrarily

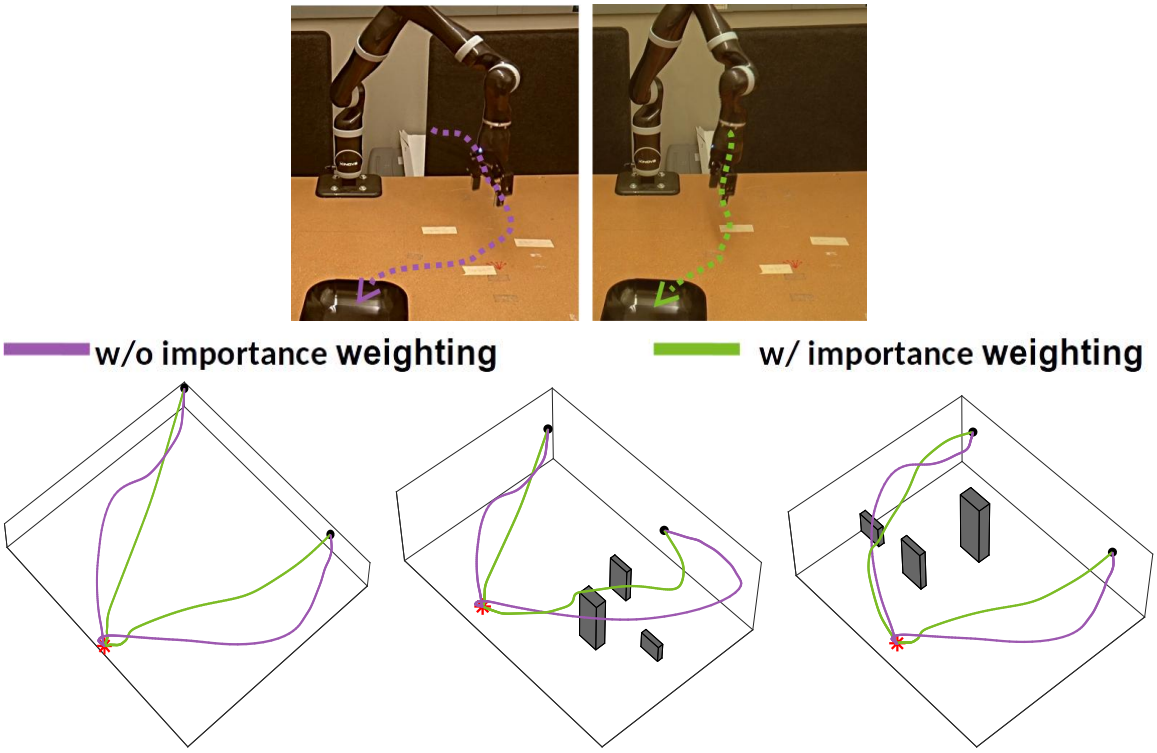


Figure 6.6: Trajectories generated by conditioning the priors on two initial positions in three different environments. *Bottom-left*: Environment without obstacles. *Bottom-center*: Environment with obstacles at the same locations as demonstrations. *Bottom-right*: Environment with obstacles displaced. *Top*: Trajectory executions on a real robot in the obstacle-free environment.

placed obstacles. We show the trajectories generated from two different initial states in three different environments. When the obstacles are placed at the same location as the demonstration phase or displaced, the reproduced trajectories from the prior without importance weighting take the longer path to the target around the obstacles. This is because the demonstrations on average took a longer path while avoiding obstacles and the prior shown in Fig. 6.5(a) forces the reproduced trajectories to exhibit a similar behavior. For the same reasons, the deviant non-smooth trajectories are also observed when no obstacles are present in the vicinity of the robot in the reproduction environment.

The *placing* skill involves placing an object at different locations on a table. All the demonstrations start from the same location since the object's initial location is fixed. The end state of the demonstration varies with the target placement location. Initially there is an obstacle present in the desired path, hence all the demonstrations go above the obstacle

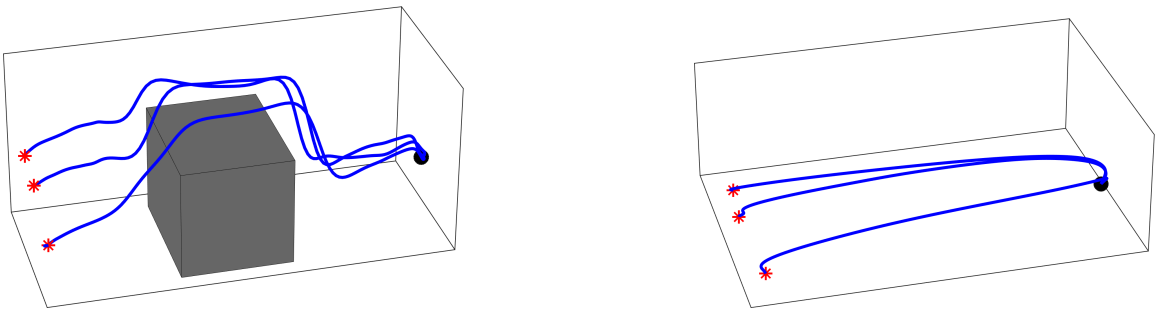


Figure 6.7: Human demonstrations for the *placing* skill in two different environments. *Left*: Environment with a large obstacle influencing the demonstrations. *Right*: Obstacle-free environment.

causing them to be influenced. Fig. 6.7 (left) plots the human demonstrations provided in this scenario. Since only the influenced demonstrations are available at this stage, the trajectory prior learned from these demonstrations also encodes the influence of obstacles which is undesirable. However, as the environment changes and more demonstrations are available in a cleaner environment, as shown in Fig. 6.7 (right), the prior is updated using the incremental weighted learning procedure described in Section 6.2.2.

Fig. 6.8 shows the evolution of the prior as demonstrations are assimilated. The prior initially enforces highly constrained motion causing the trajectories to avoid the obstacle even when it is not present. As more demonstrations are made available in an obstacle-free environment, the high importance weight relative to the influenced demonstrations enables adaptation to the desired underlying motion after just three updates. On the other hand, when the importance weighting is not considered in the incremental learning procedure, the trajectory prior still exhibits the obstacle influence even after all the demonstrations are incorporated. This is shown in Fig. 6.9. The utility of the incremental learning procedure is high in such scenarios. It is undesirable to keep all the demonstrations and re-learn the prior on arrival of each new demonstration, since this can be both time-consuming as well as memory-intensive.

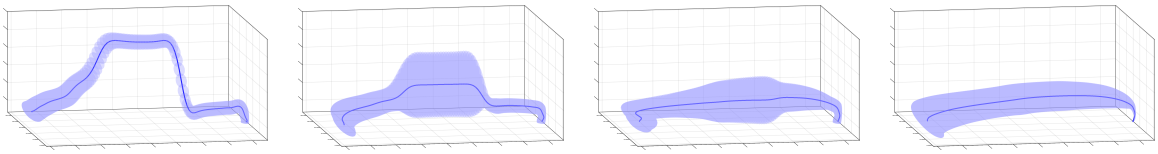


Figure 6.8: Trajectory priors for the *placing* skill with importance weighting. *Subplot 1*: Trajectory prior learned from first 3 demonstrations recorded in the presence of obstacle. *Subplots 2-3*: Prior after assimilating fourth, fifth, final demonstration respectively provided in a clean environment.

Figure 6.9: Trajectory priors for the *placing* skill without importance weighting. *Left*: Learned after assimilating first 4 demonstrations. *Right*: Final prior after all the incremental updates.

6.5 Conclusion

We have presented *importance weighted skill learning*, which is a novel technique for learning skills from demonstrations in cluttered environments and generalizing them to new scenarios. Our importance weighting function associates lower weights with parts of demonstrations that are likely to collide with obstacles. We conjecture that demonstrations which are in close proximity to obstacles are more susceptible to not satisfying the constraints of the skill being learned. Hence, those demonstrations should be given lesser importance during the skill learning stage. Our learning approach is also capable of incrementally updating and refining the skill model to incorporate new demonstrations without the need to relearn the model from scratch.

Part III

Learning Time-Invariant Skill Representations

CHAPTER 7

STABLE DYNAMICAL SYSTEMS USING EUCLIDEANIZING FLOWS

This chapter introduces our formulation of stable dynamical systems for representing reactive robot skills. In light of previously discussed methods, one may conclude that time-dependent skill representations can effectively learn and generalize a large variety of skills. Time-dependence provides the flexibility to encode fixed-horizon and speed-critical skills. Furthermore, it also allows framing skill reproduction as a trajectory optimization problem, thus enabling trajectories that are optimal and potentially feasible over its entire course. However, as discussed in Chapter 2, most time-dependent methods can neither react instantly to positional perturbations, nor can they generalize a skill beyond the time-horizon of demonstrations. Thus, time-dependent methods are not suitable for dynamic and uncertain environments. On the other hand, time-invariant dynamical systems are inherently reactive, although they do not consider optimality of the entire trajectory.

Since time-invariant dynamical systems are not limited to a fixed time duration, they can generate motions that extend over long time horizons. While this property of time-invariant dynamical systems is desirable since it makes them less susceptible to suffer from perturbations, it does carry its own set of challenges. Specifically, in the absence of any additional constraints, long-horizon rollouts from time-invariant dynamical system can diverge and blow up to infinity. To address this issue, one may impose stability constraints in the dynamical system representation, thus resulting in a *stable dynamical system*. The motions resulting from a stable dynamical system are guaranteed to remain bounded. Thus in this work, we focus on learning stable dynamical systems from human demonstrations. In our formulation, we enforce a stronger notion of stability, called *asymptotic* stability, which results in goal-directed motions i.e. motions that converge to a given target location. This is without loss of generality since complex tasks can often be achieved by an ordered

execution of goal-directed motions [95].

The challenge, however, is to encode stable human motions into dynamical systems that are stable by construction. A number of approaches have been proposed, which address this problem by explicitly parameterizing the class of stable dynamical systems and the notions of stability [9, 15, 77]. In this work, we take a fundamentally different approach than the aforementioned methods. Instead of explicitly learning a stable dynamical system, we view demonstrations as motions on a Riemannian manifold which is linked, under a smooth bijective map, i.e. a *diffeomorphism*, to a latent Euclidean space. This diffeomorphism implicitly gives rise to an inherently stable dynamical system. The learning problem thus involves finding a diffeormorphism which explains the observed demonstrations. Compared to existing diffeomorphism learning approaches to encoding motions [18, 17], our formulation, **S**table **D**ynamical **S**ystem learning using **E**uclideanizing **F**lows (SDSEF), is based on a more expressive formulation of diffeomorphisms.

We present an approach for learning a time-invariant continuous-time dynamical system (or reactive motion policy), which is globally asymptotically stable. Our dynamics formulation allows encoding severely curved goal-directed motions, and can be learned from a few demonstrations with minimal parameter tuning. Our specific contributions include: (i) a formulation of stable dynamics through warping curves into simple motions on a latent space using diffeomorphisms, and (ii) an expressive class of diffeomorphisms suitable for learning stable and smooth dynamical systems. We demonstrate the effectiveness of our approach on a standard handwriting dataset [96], and data collected on a robot manipulator [97].

7.1 Background: Stability, Diffeomorphism, and Riemannian Manifolds

We briefly summarize the theoretical background for this paper. First, we introduce the concept of global asymptotic stability and how it can be shown through an auxiliary function, i.e. a Lyapunov function. Next, we discuss how one dynamical system can be described in different ways through a change of coordinates using a diffeomorphism. Finally, we

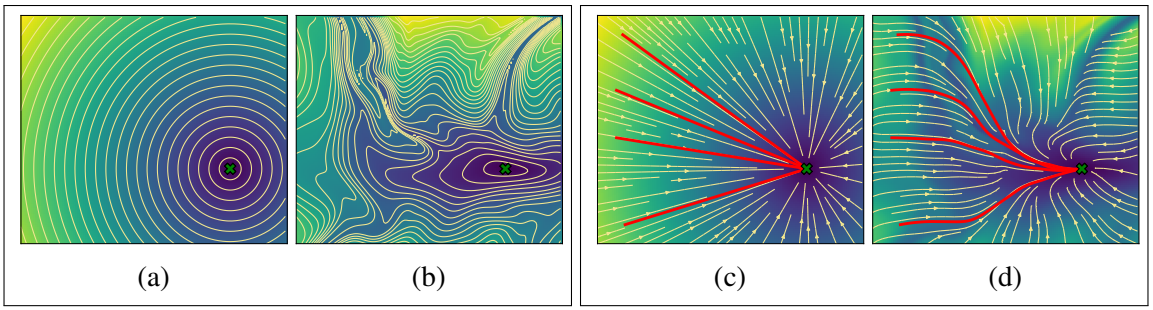


Figure 7.1: *(a)-(b)*: Isocontours showing equidistant points from the origin (green cross) in a Euclidean space and a Riemannian manifold respectively. *(c)-(d)*: Velocity fields governing shortest distance paths to the origin in the two spaces; rollouts from specific locations are overlayed in red.

shed light into the geometric interpretation of diffeomorphisms, especially when applied to gradient descent dynamical systems.

Notation: To define mappings, we use symbol $\rightarrow$ to specify domains and co-domians, e.g. $\psi : \mathbb{R}^n \rightarrow \mathbb{R}$, and $\mapsto$ for how individual elements of the domains are mapped, e.g. $\psi : \mathbf{x} \rightarrow \mathbf{y}$. We use both symbols ∇ and ∂ to denote derivatives, with a transposed relationship: consider $\mathbf{x} \in \mathbb{R}^n$ and a differentiable map $\psi : \mathbb{R}^n \rightarrow \mathbb{R}$, then $\nabla_{\mathbf{x}}\psi(\mathbf{x}) = \frac{\partial \psi}{\partial \mathbf{x}}^\top$. To evaluate functions which involve derivatives denoted by ∂, we use subscripts of brackets, e.g., $\left[\frac{\partial \psi}{\partial \mathbf{x}}\right]_{\mathbf{x}=0} = \nabla_{\mathbf{x}}\psi(0)^\top$. We use the notation $\circ$ for function composition, i.e., for $f : \mathbb{R}^m \rightarrow \mathbb{R}^p$ and $g : \mathbb{R}^n \rightarrow \mathbb{R}^m$, $(f \circ g)(\mathbf{x}) = f(g(\mathbf{x}))$.

7.1.1 Global Asymptotic Stability

Consider an n-dimensional differentiable manifold $\mathcal{M}$ with a global coordinate $\mathbf{x} : \mathcal{M} \rightarrow \mathbb{R}^n$, which maps the manifold $\mathcal{M}$ to a point in $\mathbb{R}^n$. With a slight abuse of notation, we will use the coordinate $\mathbf{x}$ to both represent the map $\mathbf{x} : \mathcal{M} \rightarrow \mathbb{R}^n$ and the point $\mathbf{x} \in \mathbb{R}^n$. A dynamical system on the manifold $\mathcal{M}$ can be described as a dynamical system on $\mathbb{R}^n$ under coordinate $\mathbf{x}$. We consider the dynamical system in the following form,

$$\dot{\mathbf{x}} = f(\mathbf{x}), \quad \text{where } f : \mathbb{R}^n \rightarrow \mathbb{R}^n \text{ is locally Lipschitz continuous.} \tag{7.1}$$

While there are multiple definitions of stability, we are specifically concerned with whether the system can converge to a point of interest $\mathbf{x}^* \in \mathbb{R}^n$ from an arbitrary initial state on

$\mathbb{R}^n$. Assume that $\mathbf{x}^*$ is an equilibrium point for the system, i.e., $f(\mathbf{x}^*) = 0$, this desired convergent property can be characterized by *global asymptotic stability*[1] of the equilibrium point $\mathbf{x}^*$. The global asymptotic stability can be shown through a continuously differentiable, positive-definite, and radially unbounded function called *Lyapunov function* [52]. A Lyapunov function $V : \mathbb{R}^n \to \mathbb{R}$ is a scalar valued function which satisfies that (i) $\dot{V}(\mathbf{x}) = \nabla_{\mathbf{x}} V(\mathbf{x})^\top f(\mathbf{x}) < 0$ for all $\mathbf{x} \neq \mathbf{x}^*$ and (ii) $\dot{V}(\mathbf{x}^*) = 0$ at the equilibrium $\mathbf{x}^*$. If there exists such a Lyapunov function V, the equilibrium point $\mathbf{x}^*$ is globally asymptotically stable: all trajectories converge to the point $\mathbf{x}^*$.

7.1.2 Change of Coordinates for Dynamical Systems

Consider a bijective map $\psi : \mathbb{R}^n \to \mathbb{R}^n$. The bijective map ψ is a *diffeomorphism* if both the map ψ and its inverse map ψ^{-1} are continuously differentiable. We further assume that the diffeomorphism ψ is bounded, namely that it maps bounded vectors to bounded vectors. A diffeomorphism can be used to describe a *change of coordinates* for differentiable manifolds. Consider again the manifold $\mathcal{M}$ with global coordinates $\mathbf{x} \in \mathbb{R}^n$. A diffeomorphism $\psi : \mathbf{x} \mapsto \mathbf{y}$ creates another global coordinate $\mathbf{y} : \mathcal{M} \to \mathbb{R}^n$ for the manifold $\mathcal{M}$ through the map $\mathbf{y} = \psi(\mathbf{x})$. As such, the dynamical system (7.1) can be described under the other coordinate $\mathbf{y}$ as,

$$\dot{\mathbf{y}} = \left[\frac{\partial \psi}{\partial \mathbf{x}} f(\mathbf{x})\right]_{\mathbf{x}=\psi^{-1}(\mathbf{y})} = \mathbf{J}_\psi(\psi^{-1}(\mathbf{y}))\, f(\psi^{-1}(\mathbf{y})) \quad := \tilde{f}(\mathbf{y}), \tag{7.2}$$

where $\mathbf{J}_\psi(\mathbf{x}) = \frac{\partial \psi}{\partial \mathbf{x}}$ is the *Jacobian matrix* of ψ. In the special case where both the system dynamics f and the diffeomorphism ψ are linear maps, this change of coordinates reduces to *change of basis* for linear dynamical systems [98].

The two dynamical systems (7.1) and (7.2) are descriptions of *the same internal dynamical system* evolving on the manifold $\mathcal{M}$. Therefore, the two systems share stability properties. Assume the existence of a Lyapunov function $V(\mathbf{x})$ showing that the equilibrium

[1] We refer the reader to [52] for a more rigorous and thorough introduction of stability properties of dynamical system.

point $\mathbf{x}^*$ is globally asymptotically stable. Then, the diffeomorphism ψ defines a Lyapunov function $\tilde{V} : \mathbf{y} \mapsto V(\psi^{-1}(\mathbf{y}))$,

$$\begin{aligned}\dot{\tilde{V}}(\mathbf{y}) &= \left[\frac{\partial V}{\partial \mathbf{x}}\,\frac{\partial \psi^{-1}}{\partial \mathbf{y}}\,\dot{\mathbf{y}}\right]_{\mathbf{x}=\psi^{-1}(\mathbf{y})} = \left[\frac{\partial V}{\partial \mathbf{x}}\,\left(\mathbf{J}_\psi(\mathbf{x})\right)^{-1}\,\tilde{f}(\mathbf{y})\right]_{\mathbf{x}=\psi^{-1}(\mathbf{y})} \\ &= \left[\frac{\partial V}{\partial \mathbf{x}}\,\left(\mathbf{J}_\psi(\mathbf{x})\right)^{-1}\mathbf{J}_\psi(\mathbf{x})\,f(\mathbf{x})\right]_{\mathbf{x}=\psi^{-1}(\mathbf{y})} = \left[\frac{\partial V}{\partial \mathbf{x}}\,\dot{\mathbf{x}}\right]_{x=\psi^{-1}(\mathbf{y})} = \dot{V}(\psi^{-1}(\mathbf{y})),\end{aligned} \tag{7.3}$$

where the second equality follows from the implicit function theorem. Therefore, the system after the change of coordinate (7.2) has a globally asymptotically stable equilibrium point $\mathbf{y}^* = \psi(\mathbf{x}^*)$. Moreover, since the diffeomorphism is bijective, the converse is also true: if there exists a Lyapunov function $\tilde{V}$ under coordinate $\mathbf{y}$, equilibrium point $\mathbf{x}^*$ is globally asymptotically stable.

7.1.3 Riemannian Manifolds and Natural Gradient Descent

From a geometric standpoint, a diffeomorphism $\psi : \mathbf{x} \mapsto \mathbf{y}$ can also help us understand the geometry of a Riemannian manifold in relation to Euclidean geometry. The geometry on an n-dimensional Riemmanian manifold can be defined by a Riemannian metric $\mathbf{G}_\psi : \mathbb{R}^n \to \mathbb{R}^{n\times n}_{++}$, which gives a notion of distance on the manifold [99].

To elaborate, let us define Euclidean goemetry on the co-domain, with coordinates $\mathbf{y}$. Consider a gradient descent dynamical system $\dot{\mathbf{y}} = \tilde{f}(\mathbf{y}) = -\nabla_{\mathbf{y}}\Phi(\mathbf{y})$ with a potential function $\Phi : \mathbb{R}^n \to \mathbb{R}$. Then, the change of coordinates defined by the diffeomorphism ψ provides a description of the same dynamics under $\mathbf{x}$ coordinate,

$$\dot{\mathbf{x}} = -\,\mathbf{G}_\psi(\mathbf{x})^{-1}\,\nabla_{\mathbf{x}}\Phi(\psi(\mathbf{x})) \quad := f_\psi(\mathbf{x}). \tag{7.4}$$

where the induced Riemannian metric in the domain is given by $\mathbf{G}_\psi(\mathbf{x}) = \mathbf{J}_\psi(\mathbf{x})^\top \mathbf{J}_\psi(\mathbf{x}) \in \mathbb{R}^{n\times n}_{++}$. The aforementioned dynamics is known as *natural gradient descent*, which is steepest descent on a Riemannian manifold [100], with respect to the potential function $\Phi \circ \psi$. The system (8.4) can generate sophisticated trajectories although the potential function Φ may only take a simple form. Moreover, if the potential function Φ is also positive definite,[2]

[2] Φ is strictly positive everywhere except $\mathbf{y}^*$, and $\Phi(\mathbf{y}^*) = 0$

convex, continuously differentiable, and radially unbounded, then the potential function Φ is a valid Lyapunov function for the gradient descent system. Therefore, both the gradient descent system and natural gradient descent system (8.4) admit a globally asymptotically stable equilibrium point.

Figure 7.1(a)–(b) shows the iso-contours of a potential function Φ which generate straight-line motions on the Euclidean space, and the corresponding potential function $\Phi \circ \psi$ on the Riemannian manifold. The isocontours show equidistant points to the goal in the Riemannian manifold, thus revealing the underlying geometry. Figure 7.1(c)–(d) shows the trajectories generated by the same underlying system observed in the corresponding coordinate systems.

7.2 Learning Stable Dynamics Using Diffeomorphisms

We view human demonstrations as goal-directed motions on an n-dimensional Riemannian manifold, governed by a stable dynamical system of the form (8.4). We view this dynamical system to be equivalent, under a change of coordinates, to another system defined on a *latent* space. Our key insight here is: *a diffeomorphism can warp a simple potential function into a more complicated one, and hence transform straight-line trajectories into severely curved motions.* As a result, the problem of learning stable dynamical systems reduces to a diffeomorphism learning problem.

7.2.1 Problem Statement

Assume the availability of N human demonstrations, each composed of T_i position-velocity pairs, denoted by $\{\{(\tilde{\mathbf{x}}_{i,t}, \dot{\tilde{\mathbf{x}}}_{i,t})\}_{t=1}^{T_i}\}_{i=1}^{N}$. We seek to find a dynamical system $\dot{\mathbf{x}} = f_\psi(\mathbf{x})$ that reproduces the demonstrations while ensuring stability. As illustrated in Section 7.1.3, we represent the system as the gradient descent system $\dot{\mathbf{y}} = -\nabla_{\mathbf{y}}\Phi(\mathbf{y})$ after a change of coordinates defined by ψ.

Although both the diffeomorphism ψ and the potential function Φ can shape the dynamical system (8.4), we view them as playing fundamentally different roles: the potential

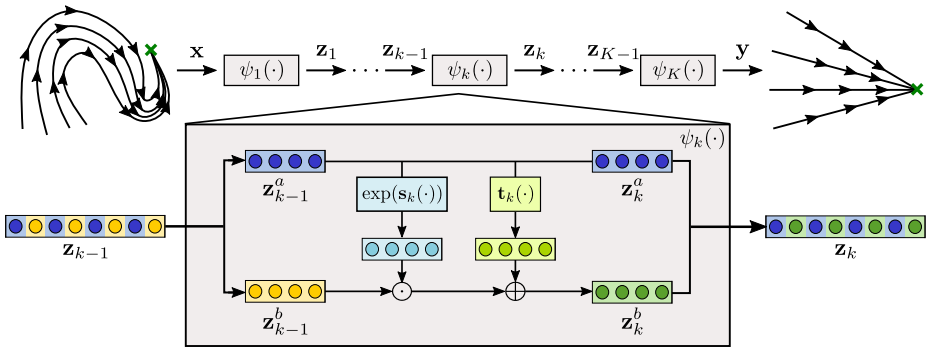


Figure 7.2: Architecture of the diffeomorphic mapping network.

function Φ dictates the theoretical property of the dynamics, e.g. stability guarantees, while the diffeomorphism ψ provides expressivity to our hypothesis class. As a result, we specify a simple potential function $\Phi(\mathbf{y}) = \|\mathbf{y} - \mathbf{y}^*\|$, with $\mathbf{y}^* = \psi(\mathbf{x}^*)$. This potential function generates unit-velocity straight-line motions to the globally asymptotically stable equilibrium point $\mathbf{y}^*$. The diffeomorphism acts to deform these straight lines to arbitrarily curved motions converging to $\mathbf{x}^*$, where the demonstrations converge. With a parameterized diffeomorphism ψ_θ, the learning problem reduces to solving,

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{\sum_{i=1}^{N} T_i} \sum_{i=1}^{N} \sum_{t=1}^{T_i} \left\| \dot{\tilde{\mathbf{x}}}_{i,t} - f_{\psi_\theta}(\tilde{\mathbf{x}}_{i,t}) \right\|_2^2 \tag{7.5}$$

To avoid notation complexity, we will drop the subscript θ in the remainder of this paper.

7.2.2 A Class of Expressive Diffeormorphisms

By definition, a diffeomorphism is required to be both, *bijective* and *continuously differentiable*. To achieve this, we impose structure in the learning problem by realizing a diffeomorphism by a composition of K diffeomorphisms $\psi = \psi_1 \circ \psi_2 \circ \cdots \circ \psi_K$, with each diffeomorphism $\psi_k : \mathbb{R}^n \to \mathbb{R}^n$ given by a *coupling layer* [59]. Let $\mathbf{z}_k \in \mathbb{R}^n$ denote the output from the kth coupling layer ψ_{k-1}, i.e., $\mathbf{z}_k = \psi_k(\mathbf{z}_k)$, $\mathbf{z}_0 = \mathbf{x}$, and $\mathbf{z}_K = \mathbf{y}$. Each coupling layer, ψ_k involves splitting the input $\mathbf{z}_{k-1}$ followed by scaling and translating one of the parts. The `split` operation divides inputs into two parts, $\mathbf{z}^a_{k-1} \in \mathbb{R}^{\lfloor n/2 \rfloor}$ and $\mathbf{z}^b_{k-1} \in \mathbb{R}^{\lceil n/2 \rceil}$, constituting alternate input dimensions, whereby the pattern of alternation is reversed after each layer. Formally,

$$\mathbf{z}_k = \begin{bmatrix} \mathbf{z}_k^a \\ \mathbf{z}_k^b \end{bmatrix} = \begin{bmatrix} \mathbf{z}_{k-1}^a \\ \mathbf{z}_{k-1}^b \odot \exp\left(s_k(\mathbf{z}_{k-1}^a)\right) + t_k(\mathbf{z}_{k-1}^a) \end{bmatrix} := \psi_k(\mathbf{z}_{k-1}), \tag{7.6}$$

where $\odot$ denotes pointwise product and $\exp$ denotes pointwise exponential. The functions $s_k : \mathbb{R}^{\lfloor n/2 \rfloor} \to \mathbb{R}^{\lceil n/2 \rceil}$ and $t_k : \mathbb{R}^{\lfloor n/2 \rfloor} \to \mathbb{R}^{\lceil n/2 \rceil}$ are the parameterized scaling and translation functions, respectively. The coupling layer is a bijective affine mapping by construction. Since each mapping is bijective, the composed mapping ψ is also guaranteed to be bijective. Further, the composed mapping ψ is continuously differentiable as long as the scaling and translation functions in each coupling layer are continuously differentiable. This formulation of a bijection has been previously employed for density estimation by Dinh et al. [59], where the scaling and translation functions were given by deep convolutional neural networks.

In contrast to [59], for the scaling and translation functions, we use single-layer neural networks with the layer resembling an approximated kernel machine [24]. This special network structure leverage the advantages of both kernel machines and neural networks: (i) the kernel functions act as regularizers to enforce desired properties on the learned function, e.g. smoothness, which can further improve the sample efficiency of the learning algorithm, and (ii) as a parameterized model, the composed diffeomorphism can be efficiently trained using learning techniques for neural networks.

We use a matrix-valued Gaussian separable kernel with length-scale l, defined as $K(\mathbf{z}, \mathbf{z}') = \exp(-\frac{\|\mathbf{z}-\mathbf{z}'\|^2}{2l^2})\mathbf{I}$. The Gaussian kernel restricts the hypothesis class to the class of C^∞ vector-valued functions, imposing a stricter smoothness constraint than continuous differentiability, i.e. C^1. This is desirable since human motions are known to be maximizing smoothness (or minimizing jerk) [101]. We approximate the aforementioned kernel by m randomly sampled Fourier features [24], such that the scaling and translation functions are given by linear combinations of these features,

$$g(\mathbf{z}) = \varphi(\mathbf{z})^\top \mathbf{w}, \qquad \text{with } \varphi(\mathbf{z}) = \sqrt{\frac{2}{m}} \begin{bmatrix} \cos(\boldsymbol{\alpha}_1^\top \mathbf{z} + \boldsymbol{\beta}_1) \\ \cos(\boldsymbol{\alpha}_2^\top \mathbf{z} + \boldsymbol{\beta}_2) \\ \vdots \\ \cos(\boldsymbol{\alpha}_m^\top \mathbf{z} + \boldsymbol{\beta}_m) \end{bmatrix} \otimes \mathbf{I} \in \mathbb{R}^{(m \cdot \lceil n/2 \rceil) \times \lceil n/2 \rceil}, \quad (7.7)$$

where $\mathbf{w} \in \mathbb{R}^{(m \cdot \lceil n/2 \rceil)}$ constitutes the learnable parameters. The projection vector $\varphi(\mathbf{z})$ is composed of m randomly sampled Fourier features such that the kernel matrix is given by $K(\mathbf{z}, \mathbf{z}') \approx \varphi(\mathbf{z})^\top \varphi(\mathbf{z}')$. Concretely, the coefficients $\{\boldsymbol{\alpha}_i\}_{i=1}^m$ are sampled from a zero-mean Gaussian distribution $\mathcal{N}(\mathbf{0}, l^{-2}\mathbf{I})$, and the bias terms $\{\boldsymbol{\beta}_i\}_{i=1}^m$ sampled from a uniform distribution $\mathrm{U}(0, 2\pi)$. Under the formulation in (7.7), we define $s_k(\cdot) = \varphi(\cdot)^\top \mathbf{w}_{s_k}$ and $t_k(\cdot) = \varphi(\cdot)^\top \mathbf{w}_{t_k}$. The set of parameters in the diffeomorphism learning problem in (7.5) is therefore given by $\theta \coloneqq \{\mathbf{w}_{s_k}, \mathbf{w}_{t_k}\}_{k=1}^K$.

7.2.3 Practical Considerations

Regarding the choice of potential functions, there are a few alternatives, including various soft versions of ℓ_2-norm, and quadratic potential functions, and other norms. In this paper, we view the potential function as dictating the stability properties while the diffeomorphisms provides expressivity to our hypothesis class. We tested the proposed ℓ_2-norm, a soft version of ℓ_2-norm, and also a quadratic potential function, and we did not observe significant differences in performance. This observation also shows that our parameterization of diffeomorphsim is expressive enough so that the choice of potential function is not significant as long as it provides desirable stability guarantees.

To solve the learning problem in (7.5) through back-propogation, the Jacobian of the diffeomorphism can be calculated analytically from (7.6) and (7.7), or through auto-differentiation packages, e.g. pytorch [102].

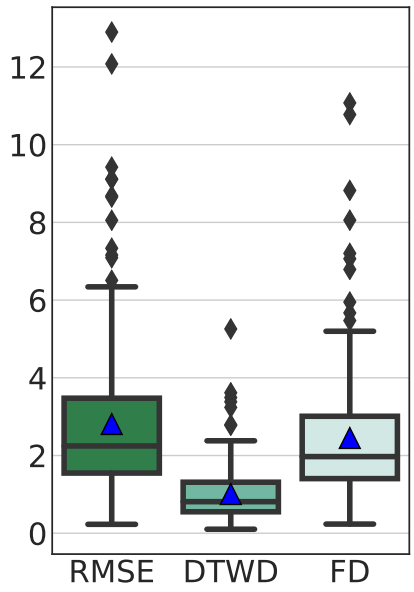


Figure 7.3: Box plots for RMSE, average DTWD, and FD, in millimeters, evaluated over the LASA dataset. The overlayed blue triangles are the means for each metric.

7.3 Experimental Results

We evaluate our approach on the LASA dataset [96] as well data collected for multiple tasks on a Franka Emika robot [97]. For all our experiments, we start the learning procedure by normalizing the demonstrations to stay in the range $[-0.5, 0.5]$. This allows us to fix the hyperparameters of our model irrespective of the scale of the data. In our experiments, we use $K = 10$ coupling layers constituting of $m = 200$ random Fourier features with length-scale $l = 0.45$. For consistency in results, the diffeomorphism is always initialized with an identity map. We optimize using ADAM [103] with default hyperparameters, alongside a learning rate of 1×10^{-4} and an ℓ_2-regularization on the weights θ with coefficient 1×10^{-8}.

The LASA dataset [96] consists of a library of 30 two-dimensional handwritten letters, each with 7 demonstrations. The scale of the dataset is $100mm \times 100mm$. For each letter, we find a diffeomorphism using (7.5). Fig. 7.5 shows the vector fields as governed by (8.4) on a subset of letters. In all the plots, the rollouts (in red) closely match the demonstrations

(in white), coming to rest at the goal. Furthermore, due to the structure imposed in learning, the dynamical system generalizes smooth and stable motions throughout the state-space. Also shown in Fig. 7.4 are isocontours of the potential function $\Phi(\psi(x))$. For quantitative evaluations, we employ three error metrics: *root mean squared error (RMSE)*, *dynamic time warping distance (DTWD)* [104], and *Frechet disance (FD)* [83]. These metrics evaluate performance of our approach in terms of its capability to reproduce the demonstrated motions. Fig. 7.3 reports these metrics, in millimeters, evaluated over 210 demonstrations (7 letters $\times$ 30 demonstrations). Each aforementioned metric focuses on different aspects of the motions. RMSE penalizes both spatial and temporal misalignment between demonstrated and reproduced motions. On the other hand, DTWD and FD disregard time misalignment, and instead focus solely on the spatial misalignment between motions. Since DTWD between any two trajectories is a time-aggregated measure of error, we report *average* DTWD, found by dividing the DTWD by the number of points T_i in a trajectory. In Fig. 7.3, the median and mean errors are observed to be small relative to the scale of the data, signifying that the learned dynamical systems are able to accurately reproduce most motions. However, higher errors are occasionally observed, accounting for outliers. This is mostly due to intersecting demonstrations which can not be modeled by a first-order dynamical system.

For demonstrations collected on a Franka Emika robot, we evaluate on two tasks: *door reaching*, and *drawer closing*. Each task dataset consists of 6 three-dimensional end-effector motions collected by physically guiding the robot. The *door reaching* task required the robot to start from inside a cabinet and reach the door handle, while the *drawer closing* task required reaching a drawer handle and pushing the drawer close. Fig. 7.6 shows the tasks, demonstrated motions (blue), as well as the reproduced motions (red). Our learning approach is observed to accurately reproduce three-dimensional motions collected on a real robot.

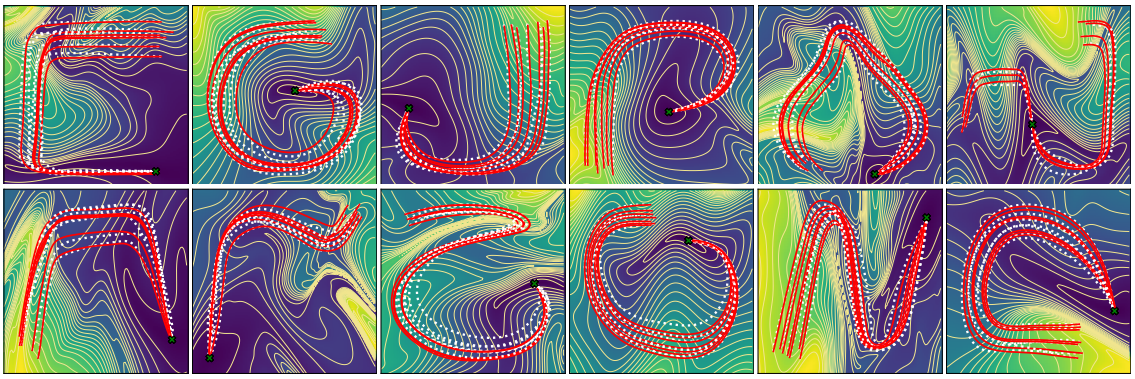


Figure 7.4: Isocontours of the potential function on LASA dataset. Overlayed are the demonstrations (white) and the reproductions (red). The reproductions cut across the isocontours.

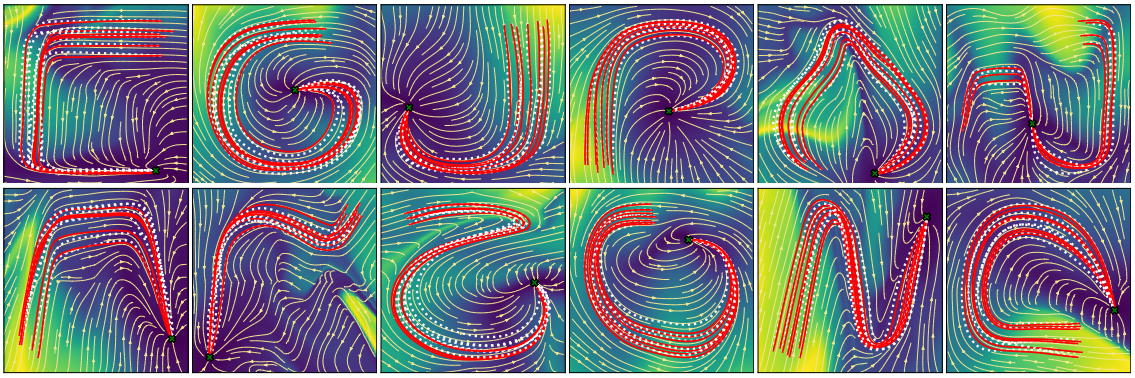


Figure 7.5: Vector fields of the dynamics learned on the LASA dataset, alongside demonstrations (white) and reproductions (red). The reproductions are governed by the natural gradient descent dynamics.

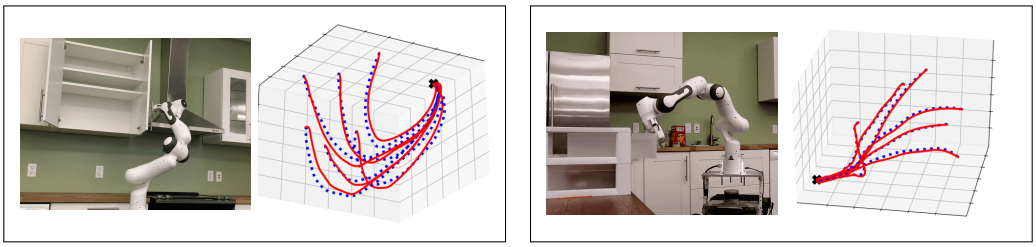


Figure 7.6: The *door reaching* (left) and *drawer closing tasks* (right). The demonstrations are plotted in blue while the reproductions are in red.

7.4 Conclusion

We have presented SDSEF, an approach for learning reactive motion skills from a few human demonstrations. SDSEF encodes complex human motions as generated from a

stable time-invariant dynamical system, linked under a learnable diffeomorphism, to a simple gradient-descent dynamical system on a latent space. A class of parameterized diffeomorphisms is proposed to learn a wide range of motions and generalize to different tasks with minimal parameter tuning. Experimental validation on a handwriting dataset and data collected on a real robot is provided to show the efficacy of the proposed approach.

CHAPTER 8

LEARNING STABLE DYNAMICAL SYSTEMS ON TRANSFORM TREES: AN END-TO-END LEARNING APPROACH

In this chapter we extend our stable dynamical system (or stable motion policy) formulation by incorporating feasibility and coordination constraints. Note that from hereon, we use the terms *dynamical system* and *motion policy* interchangeably. While these terms have broader connotations, in this work they achieve the same goal i.e. generating reactive robot motions.

Most existing works aimed at learning stable motion policies from humans [9, 18, 105], including SDSEF, assume that the motion constraints dictated by skill execution, are only associated with the end-effector of the robot. Thus, a stable dynamical system governing solely end-effector motions is typically learned. To generate corresponding configuration space commands, a tracking control law can be employed.

We argue that execution of certain complex robotic tasks may instead require simultaneous and coordinated execution of several *subtasks*, with each subtask possibly assigned to a specific robot body part. For example, consider a wiping task. In addition to requiring the end-effector to maintain contact, it may be desirable to maintain a specific elbow orientation in order to effectively wipe the surface. Movements of the elbow and the end-effector are also inter-dependent, requiring coordination. All the robot parts must also adapt to environmental changes, such as displacement of the surface to be wiped, or introduction of a new obstacle. We present an end-to-end learning approach for learning structured stable motion policies associated with coordinated and feasible execution of multiple subtasks. The class of structured motion policies or dynamical systems employed in this work, will also be referred to as tree-structured dynamical systems (TSDS).

Similar to a recent work on policy synthesis [61], TSDS parameterizes a structured motion policy by a weighted combination of several subtask policies. A subtask policy

in [61], made up of an acceleration policy and a matrix weight function (i.e. a Riemannian metric), represents motion constraints associated with a specific robot body part. The Riemannian metric dictates the relative influence of each subtask policy in the combination. The combined structured motion policy, along with its constituents, however was hand-specified in [61]. In this chapter, we instead seek to learn a similar structured motion policy from expert demonstrations. While all the constituting subtask policies can be learned, one may also choose to hand-specify a subset of them. In the latter case, the overall structured policy can viewed as made up of several parameters, some of which are fixed while others learned. Thus, the structured motion policy provides additional flexibility of embedding pre-specified behaviors (e.g. obstacle avoidance) in the learning formulation.

Instead of learning the constituting subtask policies independently, we contend that a principled way towards learning such structured motion policies should learn the constituting subtask policies together, in an end-to-end fashion. In this chapter, we show that by training the subtask policies together, we not only learn their correct relative influences, but also learn how to trade off their influences against hand-specified subtask policies. The formulation in [106] proposes an end-to-end learning framework, but is limited to rather simple subtask policies. In this chapter, we instead employ an expressive formulation of subtask policies, capable of encoding a wide range of motions. To make the resulting structured policy more amenable for learning, we reformulate the structured policy class introduced in [61] for velocity-based motion policies. As a part of this reformulation, we also introduce a recursive policy synthesis algorithm that efficiently combines the velocity-based subtask policies. Finally, we show that, under a particular choice of subtask policy class, our structured motion policy formulation guarantees stable robot motions.

In summary, we introduce an end-to-end motion policy learning technique that: (*i*) learns motions for simultaneous and coordinated execution of multiple subtasks, and (*ii*) guarantees the motions to be stable and kinematically feasible. We demonstrate the effectiveness of our learning approach on multiple robotic tasks that enforce motion constraints on several robot

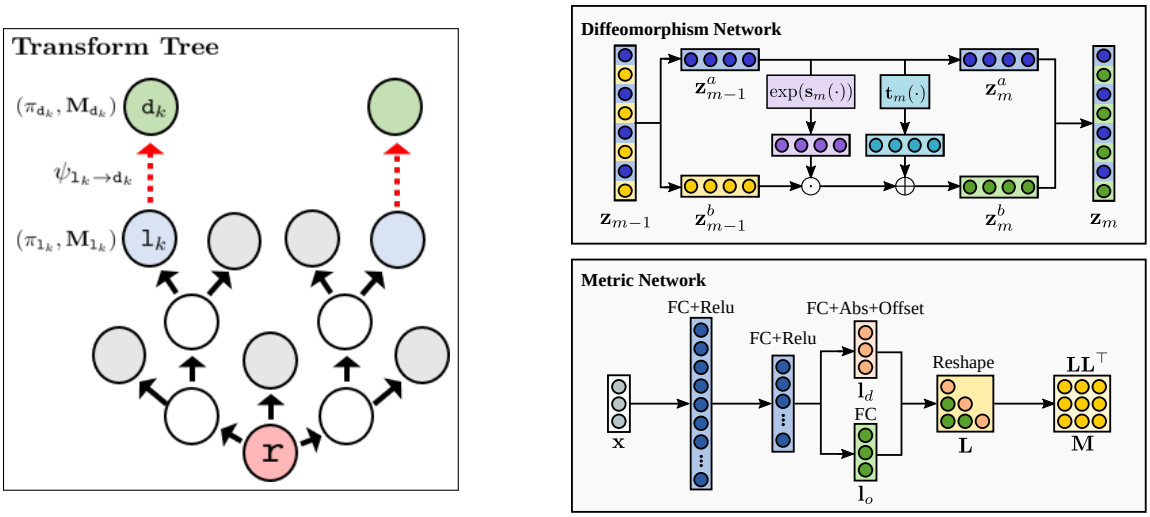


Figure 8.1: *Left:* A transform tree with root in the configuration space alongside hand-specified subtask/leaf nodes (grey) and learned subtask nodes (blue). Each learned subtask node is linked to a latent subtask node (green) under a chained map $\psi_{\mathtt{l}_k \to \mathtt{d}_k} = \psi_1 \circ \cdots \circ \psi_M$. *Top-right:* Structure of the network defining a single map ψ_m in the chain [23]. *Bottom-right:* Structure of the network for defining latent subtask metric $\mathbf{M}_{\mathtt{d}_k}$ [97].

body parts.

8.1 Motion Generation with Transform Trees

We are interested in learning policies on a decomposed task space given demonstrations in the robot configuration space. Before presenting our learning algorithm in Section 8.2, we briefly introduce the motion generation problem, which can be viewed as the dual of the learning problem.

The goal of the motion generation problem is to provide a configuration space trajectory given the desired behaviors on the task space. The robot can be tasked with multiple specifications simultaneously, which we call *subtasks*. For example, consider the task of placing an object on a shelf. The end-effector of the robot needs to reach a goal location, while also converging towards a certain approach angle. This requires coordinated movement of multiple robot body parts, each executing a subtask. Additional subtasks may also be assigned to a body part, e.g. obstacle avoidance, joint limit avoidance, or some other default behavior. A subtask can sometimes be more easily specified on its individual space, rather

than the joint configuration space. For example, collision avoidance can be described as a behavior on the 1-dimensional signed distance field. This yields a motion generation problem with subtasks defined on different *subtask spaces*.

In this section, we introduce a structure for describing the overall combined task proposed in the literature [61, 107], called a *transform tree*, as well as a computational framework for motion generation with tranform trees.

8.1.1 Transform trees

Consider a robot with its configuration space $\mathcal{C}$ given by a smooth d-dimensional manifold. We assume that the configuration space $\mathcal{C}$ admits global coordinates, called *generalized coordinates*, denoted $\mathbf{q} \in \mathbb{R}^d$. An example of generalized coordinates is the joint angles for a robot manipulator.

We assume that the overall task can be decomposed as a set of K *subtasks* defined on different *subtask spaces*, denoted $\{\mathcal{T}_k\}_{k=1}^K$. Although the maps from the configuration space $\mathcal{C}$ to each subtask space $\{\mathcal{T}_k\}_{k=1}^K$ can be viewed as independent, we consider a more general case where there is a tree structure relating the configuration space to the subtask spaces. Such a structure exists for robotic systems, e.g. those with kinematic tree structures [108]. The tree-structure can lend itself amenable to computationally efficient algorithms through reusing computations [61].

We use a *transform tree* [61] (see e.g. Fig.8.1) to describe the tree-structured map from the configuration space to subtask spaces. Each node $\mathtt{u}$ along the transform tree is associated with a manifold $\mathcal{M}$, each edge $\mathtt{e}_j$ corresponds to a smooth map $\psi_{\mathtt{e}_j} \coloneqq \psi_{\mathtt{u}\to\mathtt{v}_j}$ from the parent node manifold to the manifold associated with child node $\mathtt{v}_j$. The root node in the transform tree, $\mathtt{r}$, corresponds to the configuration space $\mathcal{C}$, and the leaf nodes $\{\mathtt{l}_k\}_{k=1}^K$ are associated with subtask spaces $\{\mathcal{T}_k\}_{k=1}^K$. We will slightly abuse the notation to denote each subtask by its associated leaf node, e.g. subtask $\mathtt{l}_k$.

8.1.2 Policy composition on transform trees

We introduce a computational framework for motion generation on transform trees, inspired by RMPflow [61]. In contrast to RMPflow, which considers acceleration policies, we are interested instead in encoding motion as a feedback velocity policy, i.e. $\dot{\mathbf{q}} = \pi(\mathbf{q})$. To make such a policy executable, we assume that the generalized velocity $\dot{\mathbf{q}}$ can be directly controlled. For robotic systems where this does not hold, such as torque-controlled robot manipulators, a low-level tracking controller [108] can be used in conjunction with the generated policy.

Consider a subtask $\mathtt{l}_k$ with a n-dimensional subtask space $\mathcal{T}_k$ with generalized coordinates $\mathbf{z}_k$. We describe the *subtask policy* by a tuple $(\pi_{\mathtt{l}_k}, \mathbf{M}_{\mathtt{l}_k})$, consisting of a nominal velocity policy $\pi_{\mathtt{l}_k} : \mathbb{R}^n \to \mathbb{R}^n$ along with a state-dependent matrix-valued importance weight, called the Riemannian metric, $\mathbf{M}_{\mathtt{l}_k} : \mathbb{R}^n \to \mathbb{R}^{n\times n}_{++}$. The Riemannian metric $\mathbf{M}_{\mathtt{l}_k}(\mathbf{z}_k)$ denotes the directional importance of the velocity policy $\pi_{\mathtt{l}_k}(\mathbf{z}_k)$ at point $\mathbf{z}_k$. For computational convenience, we also define the *momentum* policy as $\mathbf{p}_{\mathtt{l}_k} \coloneqq \mathbf{M}_{\mathtt{l}_k}\pi_{\mathtt{l}_k}$. Given the momentum $\mathbf{p}_{\mathtt{l}_k}$ and the Riemannian metric $\mathbf{M}_{\mathtt{l}_k}$, the velocity policy $\pi_{\mathtt{l}_k}$ can be uniquely recovered as $\mathbf{M}_{\mathtt{l}_k}$ is positive definite. It should be noted that the Riemannian metric $\mathbf{M}_{\mathtt{l}_k}$ and the momentum $\mathbf{p}_{\mathtt{l}_k}$ are *virtual* and do not necessarily correspond to actual physical quantities.

Given a collection of subtask policies $\{(\pi_{\mathtt{l}_k}, \mathbf{M}_{\mathtt{l}_k})\}_{k=1}^K$, the framework generates a *structured* configuration space velocity policy π which trades off approximation error to the policies $\pi_{\mathtt{l}_k}$ on each subtask space with an importance weight defined by the corresponding Riemannian metric $\mathbf{M}_{\mathtt{l}_k}$. Formally, the policy is given by the solution to the following weighted least-squares problem:

$$\pi(\mathbf{q}; \{(\pi_{\mathtt{l}_k}, \mathbf{M}_{\mathtt{l}_k})\}_{k=1}^K) \coloneqq \arg\min_{\mathbf{u}} \sum_{k=1}^K \left\| \pi_{\mathtt{l}_k}(\psi_{\mathtt{r}\to\mathtt{l}_k}(\mathbf{q})) - \mathbf{J}_{\mathtt{r}\to\mathtt{l}_k}(\mathbf{q})\,\mathbf{u} \right\|^2_{\mathbf{M}_{\mathtt{l}_k}(\psi_{\mathtt{r}\to\mathtt{l}_k}(\mathbf{q}))} \tag{8.1}$$

where $\mathbf{J}_{\mathtt{r}\to\mathtt{l}_k}$ is the Jacobian matrix of the subtask map $\psi_{\mathtt{r}\to\mathtt{l}_k}$.

The configuration space policy that solves (8.1) is given by

$$\pi = \underbrace{\left(\sum_{k=1}^{K} \mathbf{J}_{\mathtt{r}\to\mathtt{l}_k}^{\top} \mathbf{M}_{\mathtt{l}_k} \mathbf{J}_{\mathtt{r}\to\mathtt{l}_k}\right)^{\dagger}}_{\mathbf{M}_{\mathtt{r}}^{\dagger}} \underbrace{\left(\sum_{k=1}^{K} \mathbf{J}_{\mathtt{r}\to\mathtt{l}_k}^{\top} \mathbf{M}_{\mathtt{l}_k} \pi_{\mathtt{l}_k}\right)}_{\mathbf{p}_{\mathtt{r}}}. \tag{8.2}$$

As discussed in Section 8.1.1, using (8.2) for policy resolution without exploiting the structure in the task space results in redundant computations and thus introduces inefficiencies. Therefore, similar to RMPflow [61], we propose a computational framework for efficiently solving (8.1) through propagating information along the transform tree.

The algorithm consists of the following four stages:

1. *Forward pass:* From the root node to the leaf nodes, the coordinate associated with each intermediate node is calculated based on the coordinate of its parent node: $\mathbf{y}_j = \psi_{\mathtt{e}_j}(\mathbf{x})$, where $\mathbf{x}$ and $\mathbf{y}_j$ are the coordinates for the parent and the child node, respectively, and $\psi_{\mathtt{e}_j}$ is the map associated with the edge. The Jacobian matrix associated with each edge, $\mathbf{J}_{\mathtt{e}_j}$, is also evaluated.

2. *Leaf evaluation:* For each leaf node, evaluate the momentum policy $\mathbf{p}_{\mathtt{l}_k}(\mathbf{z}_k)$ and metric $\mathbf{M}_{\mathtt{l}_k}(\mathbf{z}_k)$, where the momentum policy is computed as $\mathbf{p}_{\mathtt{l}_k}(\mathbf{z}_k) = \mathbf{M}_{\mathtt{l}_k}(\mathbf{z}_k)\pi_{\mathtt{l}_k}(\mathbf{z}_k)$.

3. *Backward pass:* From the leaf nodes to the root node, recursively compute the momentum and metric at each node based the policies at the child nodes. Consider a node $\mathtt{u}$ with N child nodes $\{\mathtt{v}_j\}_{j=1}^{N}$. The momentum and metric at node $\mathtt{u}$ is calculated as,

$$\mathbf{p}_{\mathtt{u}} = \sum_{j=1}^{N} \mathbf{J}_{\mathtt{e}_j}^{\top} \mathbf{p}_{\mathtt{v}_j},\ \mathbf{M}_{\mathtt{u}} = \sum_{j=1}^{N} \mathbf{J}_{\mathtt{e}_j}^{\top} \mathbf{M}_{\mathtt{v}_j} \mathbf{J}_{\mathtt{e}_j},\ \text{where } \mathtt{e}_j \text{ is the edge from } \mathtt{u} \text{ to } \mathtt{v}_j. \tag{8.3}$$

4. *Resolve:* At the root node, the velocity policy is solved as $\pi(\mathbf{q}) = \mathbf{M}_{\mathtt{r}}^{-1} \mathbf{p}_{\mathtt{r}}$.

8.1.3 Natural gradient descent systems

The configuration space motions governed by (8.1) exhibit several desirable properties if the leaf node velocity policies on the transform tree take the form:

$$\pi_{\mathtt{l}_k}(\mathbf{z}_k) = -\mathbf{M}_{\mathtt{l}_k}^{-1}(\mathbf{z}_k)\,\nabla_{\mathbf{z}_k}\Phi_{\mathtt{l}_k}(\mathbf{z}_k), \quad \text{where } \Phi_{\mathtt{l}_k} : \mathbb{R}^n \to \mathbb{R} \text{ is called the potential function.} \tag{8.4}$$

The system (8.4) can be viewed as a continuous-time version of the *natural gradient descent* system [100], which evolves along steepest descent direction of $\Phi_{\mathtt{l}_k}$ on a Riemannian manifold defined by Riemannian metric $\mathbf{M}_{\mathtt{l}_k}$. Under the assumption that each leaf node policy is given by a natural gradient descent system (8.4), the following properties hold for the root node policy:

- *Closure:* the motion follows natural gradient descent dynamics;
- *Stability:* the system converges to the stationary points of the function

$$\Phi_{\mathtt{r}} = \textstyle\sum_{k=1}^{K} \Phi_{\mathtt{l}_k} \circ \psi_{\mathtt{r}\to\mathtt{l}_k}. \tag{8.5}$$

Formally, the above properties are stated in the following theorem:

Theorem 8.1.1. *Assume that the Riemannian metric at the root node is non-singular, i.e.* $\mathbf{M}_{\mathtt{r}} \succ 0$. *If each leaf node policy is given by a natural gradient descent dynamical system* (8.4)*, the root node policy is given by the natural descent system* $\dot{\mathbf{q}} = -\mathbf{M}_{\mathtt{r}}^{-1}\nabla_{\mathbf{q}}\Phi_{\mathtt{r}}$*, where* $\Phi_{\mathtt{r}}$ *is defined in* (8.5)*. Further, if* $\Phi_{\mathtt{r}}$ *is proper, continuously differentiable and lower bounded, the system* $\dot{\mathbf{q}} = \pi(\mathbf{q})$ *converges to a forward invariant set* $\mathcal{C}_\infty \coloneqq \{\mathbf{q} : \nabla_{\mathbf{q}}\Phi_{\mathtt{r}} = 0\}$.

Proof sketch: Assume each leaf node policy is given by natural gradient descent rule, $\mathbf{p}_{\mathtt{l}_k} = \mathbf{M}_{\mathtt{l}_k}\,\pi_{\mathtt{l}_k} = -\nabla_{\mathbf{z}_{\mathtt{l}_k}}\Phi_{\mathtt{l}_k}$, for all $k \in \{1, \ldots, K\}$. We will prove that each node follows natural gradient descent rule: Consider any non-leaf node $\mathtt{u}$. Let $\{\mathtt{v}_j\}_{j=1}^N$ be the child nodes of $\mathtt{u}$. Suppose each child node $\mathtt{v}_j$ follows natural gradient descent dynamics with potential $\Phi_{\mathtt{v}_j}$ and metric $\mathbf{M}_{\mathtt{v}_j}$. At node $\mathtt{u}$, by (8.3),

$$\mathbf{p}_{\mathtt{u}} = \textstyle\sum_{j=1}^{N} \mathbf{J}_{\mathtt{e}_j}^{\top}\,\mathbf{p}_{\mathtt{v}_j} = \sum_{j=1}^{N} \mathbf{J}_{\mathtt{e}_j}^{\top}\,\nabla_{\mathbf{y}_j}\Phi_{\mathtt{v}_j} = \nabla_{\mathbf{x}}\Phi_{\mathtt{u}}, \quad \text{where } \Phi_{\mathtt{u}} \coloneqq \textstyle\sum_{j=1}^{N} \Phi_{\mathtt{v}_j} \circ \psi_{\mathtt{e}_j} \tag{8.6}$$

Therefore, by recursively applying the analysis from the leaf nodes to the root node, we have that the root node also follows natural gradient descent dynamics $\mathbf{p}_{\mathtt{r}} = \nabla_{\mathbf{q}}\Phi_{\mathtt{r}}$. Hence,

we have,

$$\frac{d}{dt}\,\Phi_{\mathtt{r}} = \dot{\mathbf{q}}^{\top}\,\nabla_{\mathbf{q}}\Phi_{\mathtt{r}} = -\big(\nabla_{\mathbf{q}}\Phi_{\mathtt{r}}\big)^{\top}\mathbf{M}_{\mathtt{r}}^{-1}\,\nabla_{\mathbf{q}}\Phi_{\mathtt{r}} \tag{8.7}$$

Under the assumption $\mathbf{M}_{\mathtt{r}} \succ 0$, by LaSalle's invariance principle [52], the system converges to the forward invariant set $\mathcal{C}_{\infty} = \{\mathbf{q} : \nabla_{\mathbf{q}}\Phi_{\mathtt{r}} = 0\}$. □

8.2 Learning Structured Motion Policies from Demonstrations

Our goal is to synthesize a *stable* configuration space policy for *simultaneous* and *coordinated* execution of multiple subtasks. The subtasks are governed by subtask policies of the form (8.4), some of which are learned from demonstrations. While the configuration space policy is given by (8.1). In this section, we provide details of our learning approach.

8.2.1 Problem statement

Let us assume the availability of N expert trajectory demonstrations in the configuration space of the robot, each composed of T_i position-velocity pairs, denoted by $\{\{(\tilde{\mathbf{q}}_{i,t}, \dot{\tilde{\mathbf{q}}}_{i,t})\}_{t=1}^{T_i}\}_{i=1}^{N}$. Let us also assume access to a transform tree with root node $\mathtt{r}$ on the configuration space, and K leaf nodes or subtasks $\{\mathtt{l}_k\}_{k=1}^{K}$. A subset of subtasks $\{\mathtt{l}_k\}_{k=1}^{\tilde{K}}$ are learned, while the remaining $\{\mathtt{l}_k\}_{k=\tilde{K}+1}^{K}$ may be hand-specified, dictating default robot behavior. A default behavior may include avoiding obstacles, avoiding joint-limits, or reaching a target with constant velocity. We also assume that the demonstrations, when mapped to subtask spaces $\{\mathcal{T}_k\}_{k=1}^{\tilde{K}}$, are directed towards a set of unique target locations $\{\mathbf{z}_k^*\}_{k=1}^{\tilde{K}}$ where $\mathbf{z}_k^*$ is the unique target in subtask $\mathcal{T}_k$. With parameters θ defining the learnable subtask policies $\{(\pi_{\mathtt{l}_k}, \mathbf{M}_{\mathtt{l}_k})\}_{k=1}^{\tilde{K}}$, we formalize the learning problem as

$$\theta^* = \arg\min_{\theta} \frac{1}{\sum_{i=1}^{N} T_i}\frac{1}{\tilde{K}}\sum_{i=1}^{N}\sum_{t=1}^{T_i}\sum_{k=1}^{\tilde{K}} \left\|\dot{\tilde{\mathbf{q}}}_{i,t} - \pi(\tilde{\mathbf{q}}_{i,t};\theta)\right\|^2_{\mathbf{J}_{\mathtt{r}\to\mathtt{l}_k}^{\top}\mathbf{J}_{\mathtt{r}\to\mathtt{l}_k}}. \tag{8.8}$$

In the aforementioned learning problem, the norm $\|\cdot\|_{\mathbf{J}_{\mathtt{r}\to\mathtt{l}_k}^\top \mathbf{J}_{\mathtt{r}\to\mathtt{l}_k}}$ maps the target and learned configuration space velocities to the subtask space $\mathcal{T}_k$, and penalizes their mismatch. The learned configuration space velocity $\pi(\tilde{\mathbf{q}}_{i,t};\theta) \coloneqq \pi\big(\tilde{\mathbf{q}}_{i,t}\,;\theta,\{(\pi_{\mathtt{l}_k},\mathbf{M}_{\mathtt{l}_k})\}_{k=1}^K\big)$ is calculated by solving the weighted least-squares in (8.1). Note that π combines both the learned and hand-specified subtask policies. Thus, the formulation in (8.8) enables learned behavior to trade off against the default behavior as well. In the remainder of this section, we present an expressive class of stable learnable subtask policies which result in a stable configuration space velocity policy under Theorem 8.1.1.

8.2.2 A class of stable subtask policies

We seek to learn a subtask policy $(\pi_{\mathtt{l}_k},\mathbf{M}_{\mathtt{l}_k})$ defined on a subtask space $\mathcal{T}_k$ with coordinates $\mathbf{z}_k \in \mathbb{R}^n$. Furthermore, we realize the velocity policy $\pi_{\mathtt{l}_k}$ by a natural gradient descent system (8.4) given by a potential $\Phi_{\mathtt{l}_k}$ and a Riemannian metric $\mathbf{M}_{\mathtt{l}_k}$. The problem of learning a subtask policy is thus equivalent to learning a potential-metric tuple $(\Phi_{\mathtt{l}_k},\mathbf{M}_{\mathtt{l}_k})$. However, to ensure stability of the combined policy in configuration space given by (8.1), we impose structure in the parameterization of $\Phi_{\mathtt{l}_k}$ and $\mathbf{M}_{\mathtt{l}_k}$. Specifically, we require $\Phi_{\mathtt{l}_k}$ to have a unique mimima at a desired goal location $\mathbf{z}_k^*$, and $\mathbf{M}_{\mathtt{l}_k}$ to be always positive definite.

From subtasks to latent subtasks

Instead of explicitly paramterizing the subtask policy, we view a subtask policy in relation to another *latent* subtask policy. Specifically, we view a subtask space $\mathtt{l}_k$ as linked under a map $\psi_{\mathtt{l}_k\to\mathtt{d}_k} : \mathbb{R}^n \to \mathbb{R}^n$, to a latent subtask space $\mathtt{d}_k$. This results in additional edges on the transform tree (see Fig.8.1), each corresponding to $\psi_{\mathtt{l}_k\to\mathtt{d}_k}$ connecting $\mathtt{l}_k$ to $\mathtt{d}_k$. Assuming coordinates $\mathbf{x}_k \in \mathbb{R}^n$ and a natural gradient descent policy $(\Phi_{\mathtt{d}_k},\mathbf{M}_{\mathtt{d}_k})$ in the latent subtask space, the relationship in (8.3) yields,

$$\mathbf{p}_{\mathtt{l}_k} = \mathbf{J}_{\mathtt{l}_k\to\mathtt{d}_k}^\top \mathbf{p}_{\mathtt{d}_k} = -\mathbf{J}_{\mathtt{l}_k\to\mathtt{d}_k}^\top \nabla_{\mathbf{y}_k}\Phi_{\mathtt{d}_k}, \qquad \mathbf{M}_{\mathtt{l}_k} = \mathbf{J}_{\mathtt{l}_k\to\mathtt{d}_k}^\top \mathbf{M}_{\mathtt{d}_k}\,\mathbf{J}_{\mathtt{l}_k\to\mathtt{d}_k} \tag{8.9}$$

Under a similar relationship as in (8.5), the subtask space motions governed by $\pi_{\mathtt{l}_k} = \mathbf{M}_{\mathtt{l}_k}^{-1}\mathbf{p}_{\mathtt{l}_k}$, converge to the stationary points of the potential function $\Phi_{\mathtt{l}_k} = \Phi_{\mathtt{d}_k} \circ \psi_{\mathtt{l}_k \to \mathtt{d}_k}$. To ensure $\Phi_{\mathtt{l}_k}$ has a unique minima at $\mathbf{z}_k^*$, we impose additional constraints: *(i)* the latent subtask potential $\Phi_{\mathtt{d}_k}$ has a unique minima at $\mathbf{x}_k^* = \psi_{\mathtt{l}_k \to \mathtt{d}_k}(\mathbf{z}_k^*)$, and *(ii)* the latent map $\psi_{\mathtt{l}_k \to \mathtt{d}_k}$ is *diffeomorphic* i.e. continuously differentiable and bijective.. Concisely, if the latent potential has a unique minima, a diffeomorphism $\psi_{\mathtt{l}_k \to \mathtt{d}_k}$ ensures that the resulting subtask potential also has a unique minima [23].

As shown in [23], under an appropriate choice of $\psi_{\mathtt{l}_k \to \mathtt{d}_k}$, the subtask potential $\Phi_{\mathtt{l}_k}$ can generate arbitrarily curved yet stable motions, even though the latent potential $\Phi_{\mathtt{d}_k}$ may only take a simple form. Hence, we employ a simple pre-specified latent potential, e.g. $\Phi_{\mathtt{d}_k}(\mathbf{x}_k) = 0.5\|\mathbf{x}_k - \mathbf{x}_k^*\|^2$, and learn a diffeomorphism. The latent potential $\Phi_{\mathtt{d}_k}$ provides stability guarantees, while the diffeomorphism $\psi_{\mathtt{l}_k \to \mathtt{d}_k}$ provides expressivity to the linked subtask potential $\Phi_{\mathtt{l}_k}$.

Furthermore, we also learn a *latent* Riemannian metric $\mathbf{M}_{\mathtt{d}_k}$ which defines a subtask Riemannian metric $\mathbf{M}_{\mathtt{l}_k}$ by (8.9). Perhaps setting $\mathbf{M}_{\mathtt{d}_k} \equiv \mathbf{I}$ might suffice in some cases since $\psi_{\mathtt{l}_k \to \mathtt{d}_k}$ automatically induces a particular Riemannian metric in subtask space given by its jacobian inner product. However, it should be noted that $\mathbf{M}_{\mathtt{l}_k}$ plays a dual role in our formulation. Specifically, the Riemannian metric $\mathbf{M}_{\mathtt{l}_k}$ not just defines the subtask velocity policy under (8.4), but also acts as a priority weight associated with it during policy synthesis given by (8.1). Hence, we empower $\mathbf{M}_{\mathtt{l}_k}$ to correctly encode the synergies between subtasks, by also parameterizing the linked latent Riemannian metric $\mathbf{M}_{\mathtt{d}_k}$. In the proceeding subsections, we provide details of the diffeormorphism and Riemannian metric parameterizations employed in this work.

A chain of diffeomorphisms

To realize a diffeomorphism, we rely on the formulation in Chapter 7 (see Fig.8.1). Specifically, we view $\psi_{\mathtt{l}_k \to \mathtt{d}_k}$ as a chain of M simpler maps, i.e. $\psi_{\mathtt{l}_k \to \mathtt{d}_k} = \psi_1 \circ \cdots \circ \psi_M$.

Assuming coordinates $\mathbf{y}_m \in \mathbb{R}^n$ for the co-domain of ψ_m i.e. $\mathbf{y}_m = \psi_m(\mathbf{y}_{m-1})$, $\mathbf{y}_0 = \mathbf{z}_k$, and $\mathbf{y}_M = \mathbf{x}_k$, we define,

$$\mathbf{y}_m = \begin{bmatrix} \mathbf{y}_m^a \\ \mathbf{y}_m^b \end{bmatrix} = \begin{bmatrix} \mathbf{y}_{m-1}^a \\ \mathbf{y}_{m-1}^b \odot \exp\left(s_m(\mathbf{y}_{m-1}^a)\right) + t_m(\mathbf{y}_{m-1}^a) \end{bmatrix} \coloneqq \psi_m(\mathbf{y}_{m-1}), \tag{8.10}$$

where $\odot$ and $\exp$ denote pointwise product and pointwise exponential respectively. The components $\mathbf{y}_{m-1}^a \in \mathbb{R}^{\lfloor n/2 \rfloor}$ and $\mathbf{y}_{m-1}^b \in \mathbb{R}^{\lceil n/2 \rceil}$ constitute alternate input dimensions, with the pattern of alternation reversed after each mapping in the chain. Furthermore, $s_m : \mathbb{R}^{\lfloor n/2 \rfloor} \to \mathbb{R}^{\lceil n/2 \rceil}$ and $t_m : \mathbb{R}^{\lfloor n/2 \rfloor} \to \mathbb{R}^{\lceil n/2 \rceil}$ are learnable scaling and translation functions, respectively. We parameterize the scaling and translation functions as linear combinations of random Fourier features (8.10) i.e. $s_m(\cdot) \coloneqq s_m(\cdot\,; \theta_{s_m}) = \varphi(\cdot)^\top \theta_{s_m}$, and $t_m(\cdot) \coloneqq t_m(\cdot\,; \theta_{t_m}) = \varphi(\cdot)^\top \theta_{t_m}$, where the feature vector

$$\varphi(\cdot) = \sqrt{\tfrac{2}{D}}\big[\cos(\boldsymbol{\alpha}_1^\top(\cdot) + \boldsymbol{\beta}_1), \ldots, \cos(\boldsymbol{\alpha}_D^\top(\cdot) + \boldsymbol{\beta}_D)\big]^\top \otimes \mathbf{I} \quad \in \mathbb{R}^{(D\cdot\lceil n/2 \rceil)\times \lceil n/2 \rceil}, \tag{8.11}$$

is a D-dimensional Fourier feature approximation of a matrix-valued Gaussian separable kernel [109, 110], $K(\mathbf{y}, \mathbf{y}') = \exp(-\frac{\|\mathbf{y}-\mathbf{y}'\|^2}{2l^2})\mathbf{I}$ with length-scale l. Due to the choice of parameterization in (8.10)-(8.11), ψ_m is a smooth and affine bijective map, and thus a diffeomorphism. Consquently, the chain $\psi_{\mathtt{l}_k \to \mathtt{d}_k}$ is a diffeormorphism with learnable parameters $\theta_{\psi_{\mathtt{l}_k \to \mathtt{d}_k}} \coloneqq \{\theta_{s_m}, \theta_{t_m}\}_{m=1}^M$.

Riemannian metric via Cholesky decomposition

We represent a latent subtask Riemannian metric $\mathbf{M}_{\mathtt{d}_k}$ by its Cholesky decomposition parameterized by a matrix-valued neural network (see Fig.8.1). Concretely, we construct $\mathbf{M}_{\mathtt{d}_k} \coloneqq \mathbf{L}_{\mathtt{d}_k}\mathbf{L}_{\mathtt{d}_k}^\top$, where $\mathbf{L}_{\mathtt{d}_k}(\mathbf{x}_k) \in \mathbb{R}^{n\times n}$ is a lower-triangular matrix. We parameterize the vectorized diagonal and off-diagonal entries of $\mathbf{L}_{\mathtt{d}_k}$, i.e. $\mathbf{l}_d(\mathbf{x}_k; \theta_{\mathbf{l}_d}) \in \mathbb{R}^n$ and $\mathbf{l}_o(\mathbf{x}_k; \theta_{\mathbf{l}_o}) \in \mathbb{R}^{\frac{1}{2}(n^2-n)}$ respectively, as fully-connected neural networks with RELU activa-

tions. Furthermore, the networks for $\mathtt{l}_o$ and $\mathtt{l}_d$ share parameters for all the layers except their output layers. To ensure $\mathbf{L}_{\mathtt{d}_k}$ is a valid Cholesky decomposition, and consequently $\mathbf{M}_{\mathtt{d}_k}$ is positive definite, we require the entries of $\mathtt{l}_d$ to be strictly positive. In lieu of this, we take the absolute value of the output linear layer of $\mathtt{l}_d$ and add a small positive bias $\epsilon > 0$. The parameters of the latent Riemannian metric $\mathbf{M}_{\mathtt{d}_k}$ are concisely denoted by $\theta_{\mathbf{M}_{\mathtt{d}_k}} \coloneqq \{\theta_{\mathtt{l}_d}, \theta_{\mathtt{l}_o}\}$

In line with the aforementioned formulations, the parameters in (8.8) corresponding to the learnable subtask policies are thus given by $\theta \coloneqq \{\theta_{\psi_{\mathtt{l}_k \to \mathtt{d}_k}}, \theta_{\mathbf{M}_{\mathtt{d}_k}}\}_{k=1}^{\tilde{K}}$.

8.3 Experimental Results

We evaluated our approach on three manipulations tasks[1] on a 7-DOF Rethink Sawyer robot with configuration space coordinates $\mathbf{q} \in \mathbb{R}^7$. Each of the tasks is decomposed into subtasks assigned to 3 robot body parts, whereby each body part is represented by a unique *control point* (see Fig. 8.2 for details). Given our choice of control points, the subtask policies effectively control the end-effector pose (i.e. position and orientation). However, we stress that our learning approach is not only limited to learning robot poses. In fact, one may instead, for instance, choose to learn motion policies dictating a partial pose (by removing a control point), or pose alongside the robot elbow (by adding an additional control point). Furthermore, as shown in our experiments, our framework enables learning subtask policies in combination with other pre-specified subtask policies.

As a prelude to learning, we setup a transform tree (see Section 8.1.1). Specifically, the tree had its root in the robot's configuration space, and leaves associated with the subtask spaces mapped by $\psi_{\mathtt{r} \to \mathtt{l}_k}$. To elaborate, each leaf node $\mathtt{l}_k$ had a parent node associated with one of the control points, which were further linked to the root node under the robot's kinematic chain. In the absence of any learned behaviors, the robot would execute default behavior dictated by the 3 default subtask policies. The default subtask policies, governed by a convex potential and a constant Riemannian metric $\mathbf{M}_{\mathtt{l}_k} = 10\mathbf{I}$, pull the end-effector

[1] Video of experiments: https://youtu.be/hwcxzLnxZPQ

in straight-line towards a desired goal pose. Additionally, to ensure the root Riemannian metric $\mathbf{M}_{\mathtt{r}}$ is always non-singular, we added a small offset $\epsilon_{\mathtt{r}} = 0.02$ to its diagonal entries.

We consider 3 tasks including *inspection*, *placing-1*, and *placing-2*. For details about the task specifications, the reader is referred to Figs. 8.4–8.6. For each task, a human subject provided multiple configuration space demonstrations via kinesthetic teaching, i.e. 14 demonstrations for *inspection*, 9 for *placing-1*, and 12 for *placing-2*. To bias the robot's motions towards a demonstrated behavior, we added additional $\tilde{K} = 3$ leaves to the tree, whereby each new leaf also branched out from one of the control point nodes. Overall, the tree consisted of $K = 6$ leaves. We learned the subtask policies $\{(\pi_{\mathtt{l}_k}, \mathbf{M}_{\mathtt{l}_k})\}_{k=1}^{\tilde{K}=3}$ as per Section 8.2, defined by a set of diffeomorphisms $\{\psi_{\mathtt{l}_k \to \mathtt{d}_k}\}_{k=1}^{\tilde{K}}$ and a set of latent Riemannian metrics $\{\mathbf{M}_{\mathtt{d}_k}\}_{k=1}^{\tilde{K}}$. Furthermore, each diffeomorphism $\psi_{\mathtt{l}_k \to \mathtt{d}_k}$ was composed of $M = 10$ chained diffeomorphisms, each parameterized by $D = 200$ random Fourier features with length-scale $l = 0.45$. On the other hand, each latent Riemannian metric $\mathbf{M}_{\mathtt{d}_k}$ had two hidden layers with 128 and 64 dimensions respectively. The optimization problem in (8.8) was solved with Adam optimizer [103] with a learning rate of 1×10^{-4} and weight decay 1×10^{-8}.

To evaluate the performance of our approach, we establish two baselines: *(i)* an *independent* learning version whereby the subtask policies are learned independently, and *(ii)* a *single link* learning version where just a single control point (i.e. end-effector) is chosen and the associated subtask policy is again learned independently. Fig. 8.2 shows example reproductions of end-effector pose trajectories under the aforementioned variants of our algorithm. Our coordinated learning approach is observed to successfully reproduce the demonstrated motions. However, the baselines either fail to reproduce the position profile or the orientations. To quantitatively evaluate the capacity of our approach to reproduce demonstrations, we employ two error metrics i.e. mean position error, and mean orientation error. We evaluate position errors in terms of Euclidean distance i.e. $error(\boldsymbol{p}_1, \boldsymbol{p_2}) = \|\boldsymbol{p}_1 - \boldsymbol{p}_2\|_2$, where p_1 and p_2 are adjacent (in terms of time) end-effector positions on the demonstrated

and reproduced trajectory respectively. On the other hand, for orientation error we employ $error(\boldsymbol{o}_1, \boldsymbol{o}_2) = \arccos(|\boldsymbol{o}_1 \cdot \boldsymbol{o}_2|)$, where $\boldsymbol{o_1}$ and $\boldsymbol{o}_2$ are unit quaternions representing end-effector orientations. For each comparison metric, we take the mean of the errors accumulated over the duration of a trajectory. Fig. 8.3 reports these comparisons as box plots. For the two *placing* tasks, our approach outperforms the baselines by a significant margin. A major contributor towards this difference in performance is the existence of default subtask policies. When learned without accounting for the existing policies, the learned policies may not be able to sufficiently bias against the default behavior. Furthermore, we also observe that the independent learning version occasionally performs worse than the single link learning variant. This is perhaps because the independently learned subtask policies may conflict with each other too. This does not manifest as much in the single link case, since there is only one learned subtask policy.

Lastly, we also test the generalization performance of our approach. For this evaluation, we execute rollouts of our motion policy from 10 novel initial configurations. We considered a rollout as successful if all the goals of the task were met without any collisions. Fig. 8.3(*right*) reports the success rates. Once again, our end-to-end learning approach is seen to outperform the baselines in terms of generalization success rates. We also observed that while the difference in terms of quantitative errors between our approach and the baselines was small on the *inspection* task, there were vast differences in performances given by generalization success rates. This is perhaps because, even when not trained end-to-end, the robot's kinematic constraints may enforce certain level of coordination between subtasks, thus resulting in low reproduction errors. However, for highly constrained tasks like the ones we explore in this chapter, even small errors can result in task execution failures.

Rollouts from our learned motion policies, starting from the same configurations as demonstrations, are also visualized in Figs. 8.4–8.6 (*bottom*). While we validated our approach on all the demonstrations, only a subset of rollouts is visualized here.

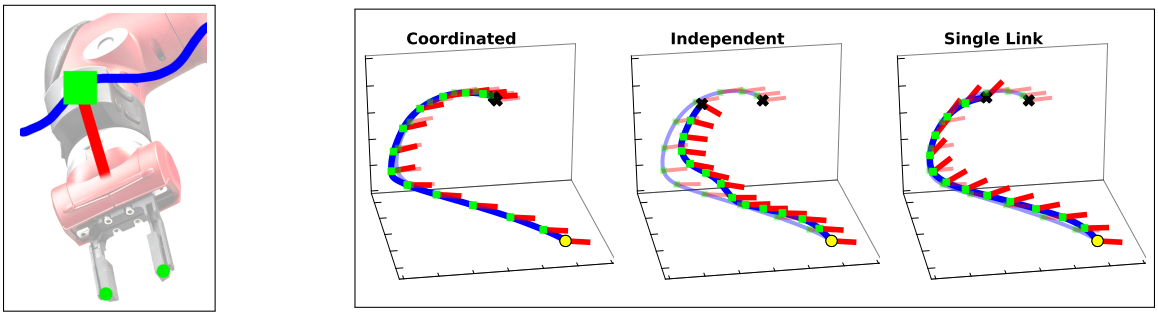


Figure 8.2: *Left*: Visualization of the 3 control points (in green), with the end-effector control point denoted by a square while the two control points for gripper tips are given by circles. Overlaid is an end-effector position trajectory (in blue), and a line directed from the end-effector to the center of the gripper (in red) denoting instantaenous end-effector orientation. *Right*: Plots showing example pose trajectories starting from an initial end-effector pose (yellow circle) governed by our approach and the baselines. Also shown in the background, is the demonstration starting from the same initial pose. The final positions are denoted by black crosses.

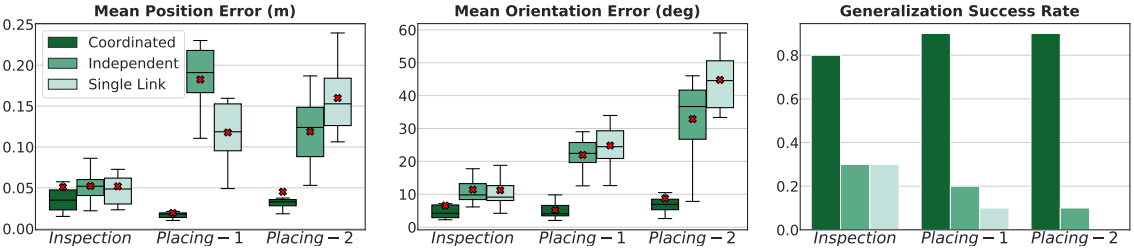


Figure 8.3: Performance comparison of our approach against the baselines based on, mean position error (*left*), mean orientation error (*center*), and generalization success rate over 10 executions (*right*).

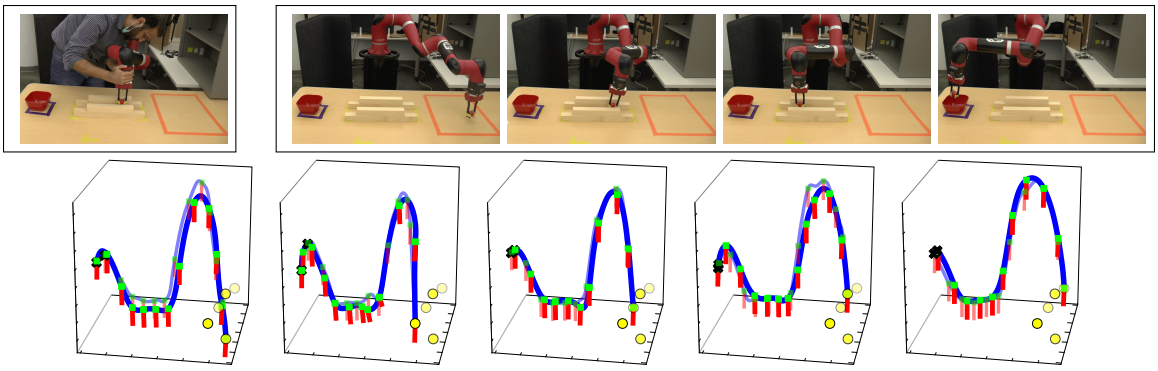


Figure 8.4: The *inspection* task required the robot to pick an object from one side of the table and place it in a bowl on the other side. In the middle, the robot was required to pass a constrained pathway. *Top-left:* a human demonstration. *Top-right:* A series of snapshots showing a robot executing learned behavior. *Bottom:* Plots of a subset of motion reproductions from different initial poses, overlayed on corresponding demonstrations. The yellow circles represent the initial end-effector positions, each corresponding to one of the rollouts.

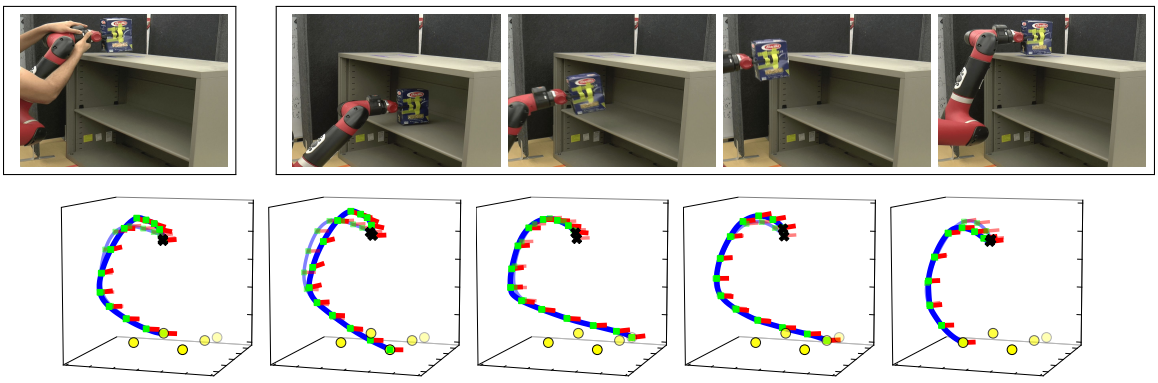


Figure 8.5: The *placing-1* task required the robot pick an object from a lower shelf and place it on at a goal location on the top-most shelf at a certain orientation. *Top-left:* a human demonstration. *Top-right:* A series of snapshots showing a robot executing learned behavior. *Bottom:* Plots of a subset of motion reproductions from different initial poses, overlaid on corresponding demonstrations. The yellow circles represent the initial end-effector positions, each corresponding to one of the rollouts.

8.4 Conclusion

We introduced an end-to-end learning framework for learning simultaneous, coordinated and stable motions from demonstrations. The motions are governed by a structured motion policy class, resulting from decomposing a a motion policy in configuration space into several subtask policies setup on a transform tree. The underlying structure additionally allows trading off learned behaviors against pre-specified default behaviors, such that

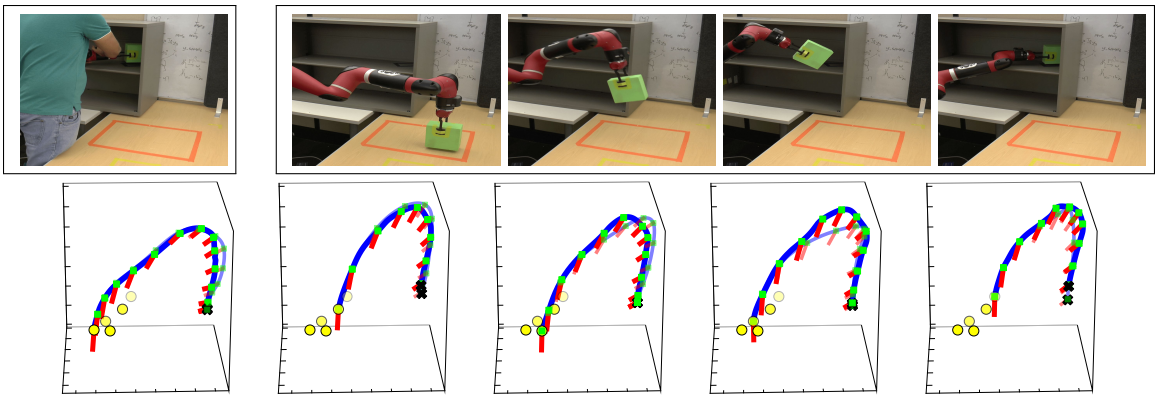


Figure 8.6: The *placing-2* task required the robot pick an object from a table, significantly rotate its end-effector, and place the object on a shelf. *Top-left:* a human demonstration. *Top-right:* A series of snapshots showing a robot executing learned behavior. *Bottom:* Plots of a subset of motion reproductions from different initial poses, overlaid on corresponding demonstrations. The yellow circles represent the initial end-effector positions, each corresponding to one of the rollouts. Note that the viewing angle in the plots is different from that in the robot execution snapshots.

the combined behavior matches with the expert. We supplement our formulation with experiments conducted on a robot manipulator executing constrained manipulation tasks.

CHAPTER 9

AN ALTERNATIVE FORMULATION FOR LEARNING STRUCTURED MOTION POLICIES: LEARNING SUBTASK POLICIES INDEPENDENTLY

In this chapter, we present an alternative learning formulation geared towards structured motion policies presented in the previous chapter. Specifically, instead of end-to-end learning of all the constituent subtask policies, we present a formulation for learning the subtask policies independently. It should be noted here that although the end-to-end learning formulation is more principled, we present the preceding independent learning procedure for the sake of completeness.

Our premise here is that even when the subtask policies are learned independently, the kinematic chain of the robot still enforces a certain level of coordination between these subtask policies. However, there are two key assumption in this formulation. First, the subtask policies are assumed to be always compatible with each other. Since the state-dependent weights associated with subtask policies, are not learned together, they are incapable of carrying out policy resolution in case conflicts occur. Thus for the tasks we consider in this chapter, the subtasks are always compatible since they are related by the kinematics of the robot. Second assumption in this formulation is regarding non-existence of hand-specified policies on the task map tree. Even if hand-specified policies do exist, their influence (given by their state-dependent weight) is negligible in comparison to the learned subtask policies. When this assumption breaks, the combined structured policy may not generalize in a desirable manner, as observed in Section 8.3 of the previous chapter.

Unlike Chapter 8, in this chapter we employ a second-order representation of subtask motion policies, called *Riemannian Motion Policies* (RMPs) [107]. An RMP is a mathematical object composed of an acceleration policy along with a Riemannian metric[1], which

[1]The Riemannian metric also functions as an inertia matrix, and thus the two terms are used interchangeably.

defines the underlying non-Euclidean geometry of the subtask space. We learn multiple subtask RMPs independently, from human demonstrations. To combine the subtask RMPs to generate a structured configuration space policy, we utilize RMPflow [61]. RMPflow is the second-order equivalent of our policy synthesis formulation described in Section 8.1.2. We emphasize here that while we employ a second-order (i.e. acceleration-based) representation in this chapter, a first-order (i.e. velocity-based) representation, similar to that in Chapter 8, can also be used instead.

We demonstrate the effectiveness of the proposed independent learning approach on *door reaching* and *drawer closing* tasks.

9.1 Background: Riemannian Motion Policies

Consider a robot with its configuration space $\mathcal{C}$ given by a smooth manifold, admitting a global generalized coordinate $\mathbf{q} \in \mathbb{R}^d$. Oftentimes, it is more convenient to describe the specifications of the robot motion on another manifold called the *task space*, denoted $\mathcal{T}$, where there exists a smooth task map $\psi : \mathcal{C} \to \mathcal{T}$ that maps the configuration space to the task space. The RMP framework [61, 107] assumes that the overall task can be decomposed into a set of *subtasks* defined on different *subtask spaces*. Examples of subtasks include goal reaching or trajectory following for the end-effector, collision avoidance for a certian part of the robot, etc. The task space $\mathcal{T}$ can thereby be represented as the collection of multiple *subtask* spaces. The goal of RMPs and RMPflow [107, 61] is to generate an acceleration policy $\mathbf{a_r} = \pi(\mathbf{q}, \dot{\mathbf{q}})$ on the configuration space $\mathcal{C}$ such that the transformed policies exhibit the desired behaviors in each of the subtask spaces.

9.1.1 Riemannian Motion Policies (RMPs)

A Riemannian Motion Policy (RMP) [61, 107] is a mathematical object describing motions on manifolds. Consider an m-dimensional manifold $\mathcal{M}$ with generalized coordinates $\mathbf{x} \in \mathbb{R}^m$. An RMP on the manifold $\mathcal{M}$ can be succinctly represented by its *canonical form* as a pair $(\mathbf{a}, \mathbf{M})^{\mathcal{M}}$. Here, the first component, $\mathbf{a} \in \mathbb{R}^m$ is an acceleration policy governing

motions on the manifold, while the second component, $\mathbf{M} \in \mathbb{R}_{+}^{m \times m}$ is a Riemannian metric, that is a positive definite matrix defining the structure of the underlying manifold. An alternative parameterization of RMP which will become useful later is given by the *natural form* $[\mathbf{f}, \mathbf{M}]^{\mathcal{M}}$ where $\mathbf{f} \coloneqq \mathbf{M}\,\mathbf{a}$ is the desired *force map*.

One realization of an RMP is a *virtual* dynamical system [61],

$$\mathbf{a} = \mathbf{M}(\mathbf{x})^{-1}\big(- \nabla \Phi(\mathbf{x}) - \mathbf{B}(\mathbf{x})\,\dot{\mathbf{x}} - \boldsymbol{\xi}_{\mathbf{M}}(\mathbf{x}, \dot{\mathbf{x}})\big), \tag{9.1}$$

where, as in Geometric Mechanics [111], the Riemannian metric $\mathbf{M}(\mathbf{x}) : \mathbb{R}^m \to \mathbb{R}_{+}^{m \times m}$ serves as the *inertia matrix*, $\mathbf{B}(\mathbf{x}) : \mathbb{R}^m \to \mathbb{R}_{+}^{m \times m}$ is the *damping matrix* and $\Phi(\mathbf{x}) : \mathbb{R}^m \to \mathbb{R}_{+}$ the *potential function*. The system in (9.1) dictates the motion on a manifold under the influence of a damped virtual potential field. An important property of this system is that it is inherently *Lyapunov stable*.

The Riemannian metric $\mathbf{M}$ also induces the curvature term $\boldsymbol{\xi}_{\mathbf{M}}$. The ith component of the curvature term, denoted $(\xi_{\mathbf{M}})_i$, is given by

$$(\xi_{\mathbf{M}})_i = \sum_{j=1}^{n} \sum_{k=1}^{n} \frac{1}{2} \left(\frac{\partial\, M_{ij}(\mathbf{x})}{\partial\, x_k} + \frac{\partial M_{ki}(\mathbf{x})}{\partial\, x_j} - \frac{\partial\, M_{jk}(\mathbf{x})}{\partial\, x_i} \right) \dot{x}_j\, \dot{x}_k, \tag{9.2}$$

where we use M_{ij} to denote an entry of a matrix $\mathbf{M}$ and x_i to denote a component of a vector $\mathbf{x}$. Intuitively, the curvature term $\boldsymbol{\xi}_{\mathbf{M}}$ bends the trajectories to follow geodesics on the manifold in the absence of the potential and damping terms. It should be noted that the Riemannian metric $\mathbf{M}(\mathbf{x})$ employed in this chapter is only position dependent. In the case when the Riemannian metric is both position and velocity-dependent, a generalization of the system (9.1), called Geometric Dynamical System (GDS) [61], can be used instead.

For simple tasks like reaching a goal without any additional specifications, a system that generates the RMP can be hand-specified as is done in [107]. However, for more complex behaviors, it can be hard or sometimes infeasible for an end-user to design RMPs. To mitigate this problem, in Section 9.2 we detail a method which can instead learn RMPs of

the form (9.1) from human demonstrations.

9.1.2 RMPflow

RMPflow [61] is a computational framework to generate reactive policies by combining RMPs. Given multiple individually specified (or learned in the subsequent sections) RMPs for different subtasks, RMPflow combines these policies into one global configuration space policy.

The core data structure of RMPflow is the RMP-tree[2], a directed tree encoding the structure of the task map. Specifically, each node $\mathtt{v}$ along the RMP-tree is made up of a state $(\mathbf{x}, \dot{\mathbf{x}})$ on a manifold along with an associated RMP $(\mathbf{a}_{\mathtt{v}}, \mathbf{M}_{\mathtt{v}})^{\mathcal{M}}$. Each edge $\mathtt{e}$ in the RMP-tree corresponds to a smooth map $\psi_{\mathtt{e}}$ from the given parent node manifold to the child node manifold. The root node in the RMP-tree, $\mathtt{r}$, is associated with the state $(\mathbf{q}, \dot{\mathbf{q}})$ in the configuration space $\mathcal{C}$ and its policy $(\mathbf{a}_{\mathtt{r}}, \mathbf{M}_{\mathtt{r}})^{\mathcal{C}}$. Let K be the number of leaf nodes in the RMP-tree. The leaf nodes $\{\mathtt{l}_k\}_{k=1}^{K}$ are associated with subtask RMPs $\{(\mathbf{a}_{\mathtt{l}_k}, \mathbf{M}_{\mathtt{l}_k})^{\mathcal{T}_{\mathtt{l}_k}}\}_{k=1}^{K}$. Each subtask RMP encodes a desired subtask acceleration policy, while the associated Riemannian metric assigns a state-dependent importance weight to the policy when combined with other policies. An example RMP-tree is shown in Figure 9.1a. Recursive application of RMP-algebra (cf. Appendix A.1) along the RMP-tree enables a weighted combination of the leaf node RMPs to generate a global configuration space policy $\mathbf{a}_{\mathtt{r}} = \pi(\mathbf{q}, \dot{\mathbf{q}})$.

One important feature of RMPflow is that it preserves the stability property of leaf node policies: if all subtask RMPs are generated by systems in the form of (9.1), the combined policy in the configuration space is also in the form of (9.1) and hence Lyapunov stable. In the following section, we will make full use of the stability property of RMPflow to learn RMPs that can be combined (with both hand-specified RMPs and learned RMPs) into a stable policy.

[2]The RMP-tree is equivalent to the transform tree presented in Section 8.1.1

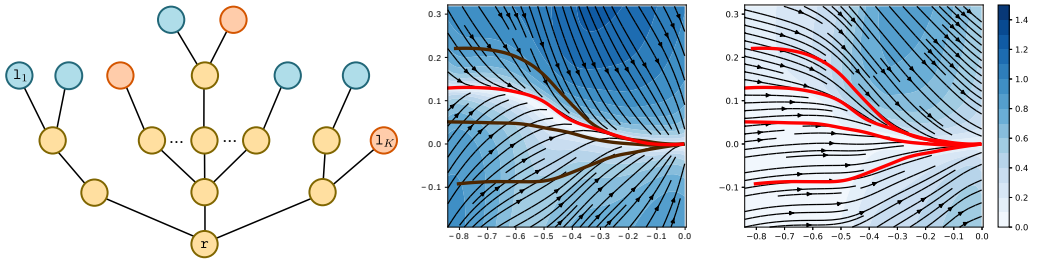
9.2 Skill Reproduction via RMPflow

For robot manipulators, a skill may involve coordinated and constrained motion of different parts of a robot. To encode a skill, we first construct an RMP-tree with root node in the configuration space, specifically the joint space of the robot. The relevant robot body parts are added as child nodes of the root in the RMP-tree with edges given by the forward kinematics of the robot. Branching out further from these nodes are leaf nodes, corresponding to various subtask spaces. We propose to learn leaf node RMPs from human demonstrations. A human can choose to provide demonstrations for each subtask space either independently or simultaneously. Additional hand-specified leaf RMPs, for example obstacle avoidance and joint-limit RMPs [61] can be added along the RMP-tree to ensure feasibility of robot motions. The overall configuration space policy $\mathbf{a_r}$ is found using RMPflow as described previously. It should be noted here that in order to ensure stability of the configuration space policy, all RMPs employed in this work are of the form (9.1).

9.2.1 Human-Guided Riemannian Motion Policies

In the kth subtask space $\mathcal{T}_{\mathtt{l}_k}$, corresponding to leaf node $\mathtt{l}_k$, our aim is to learn an RMP of the form (9.1). In lieu of this, we assume the availability of N human demonstrations $\{\boldsymbol{\zeta}_i^{(k)}\}_{i=1}^{N}$. Here the ith trajectory demonstration is composed of T_i datapoints $\boldsymbol{\zeta}_i^{(k)} = \{\boldsymbol{\zeta}_{i,t}^{(k)}\}_{t=0}^{T_i}$, and all subtask space trajectories converge and come to rest at a common target position $\boldsymbol{\zeta}_{i,T_i}^{(k)} = \boldsymbol{\zeta}_T^{(k)}$. For notational clarity, we will drop the subtask space subscript k in the remainder of this section. However we will always refer to a particular subtask unless otherwise stated.

We observe here that the desired motion generated by the system (9.1) is mainly governed by the component $-\mathbf{M}(\mathbf{x})^{-1}\nabla\Phi(\mathbf{x})$, which can be seen as setting $\ddot{\mathbf{x}}^d$ along the negative *natural gradient* of the potential function on the manifold. The remaining components: the *damping acceleration* $-\mathbf{M}(\mathbf{x})^{-1}\,\mathbf{B}(\mathbf{x})\,\dot{\mathbf{x}}$ and the *curvature acceleration* $-\mathbf{M}(\mathbf{x})^{-1}\,\boldsymbol{\xi}_{\mathbf{M}}(\mathbf{x},\dot{\mathbf{x}})$, simply ensure stable and geometrically consistent behavior. Hence, we view the learning from demonstration problem as equivalent to learning the aforementioned natural gradient



(a) An example RMP-tree (b) Potential function (c) Potential w/ learned metric

Figure 9.1: (a) An example RMP-tree: both learned leaf node policies (blue) and hand specified policies can be combined to generate a global (root node) policy at the configuration space. (b) Vector field introduced by the learned potential function: every point in the trajectory are attracted to the single demonstration (red) with which the potential function is learned. (c) Vector field generated by the learned potential and metric: the space is warped by the learned metric so that the demonstrations can be reproduced.

descent subsystem. The damping component is pre-specified as $\mathbf{B}(\mathbf{x}) = \gamma_d \mathbf{M}(\mathbf{x})$, such that the damping acceleration $\mathbf{M}(\mathbf{x})^{-1}\mathbf{B}(\mathbf{x}) \equiv \gamma_d \mathbf{I}$ is always independent of the choice of metric.

The problem of learning the natural gradient descent subsystem is highly under-constrained, that is, there can be many different potential and metric combinations that can result in the same desired acceleration policy. We choose to bias the solution of this learning problem via a two-step process. Specifically, we first learn a less expressive potential function and then learn a Riemannian metric which warps the gradient of the potential such that the overall policy is expressive enough to accurately reproduce the demonstrations. According to [61], for $\mathbf{M}(\mathbf{x}), \mathbf{B}(\mathbf{x}) \in \mathbb{R}_{+}^{m \times m}$, the system in (9.1) converges to the forward invariant set $\mathcal{C}_\infty := \{(\mathbf{x}, \dot{\mathbf{x}}) : \nabla\Phi(\mathbf{x}) = 0, \dot{\mathbf{x}} = 0\}$ [61]. Hence, we require our learned potential to have a minima at the target and our learned Riemannian metric to be globally positive definite. The remainder of this section details our learning procedure.

9.2.2 Learning Nominal Potential from Demonstrations

In order to learn a subtask space potential function, we first select a nominal demonstration $\boldsymbol{\zeta}^*$ which is the least dissimilar from the other demonstrations in terms of geometric features. One such metric of dissimilarity is Dynamic Time Warping (DTW) distance. The nominal

demonstration is therefore given by the demonstration with the least mean DTW distance from other demonstrations.

Given the nominal trajectory demonstration $\boldsymbol{\zeta}^*$, we learn a potential $\Phi(\mathbf{x})$, which generates a dissipative field $-\nabla\Phi(\mathbf{x})$ that produces motions which: *(i)* converge smoothly towards the nominal trajectory, and *(ii)* follow the remaining trajectory after convergence. Furthermore, to ensure stability we also enforce

$$\nabla\Phi(\boldsymbol{\zeta}_T) = 0, \qquad \Phi(\mathbf{x}) \to \infty \text{ as } \mathbf{x} \to \infty, \tag{9.3}$$

so that the trajectories can always be bounded following the dynamics (9.1). Figure 9.1b shows a vector field generated by the negative gradient of an example potential field with aforementioned properties. We use an approach similar to that used by [112]. Specifically, the overall potential is given by a convex combination of potential elements centered at each trajectory point $\boldsymbol{\zeta}_t^*$,

$$\Phi(\mathbf{x}) = \sum_{t=1}^{t=T} w_t(\mathbf{x})\phi_t(\mathbf{x}), \quad \text{where } w_t(\mathbf{x}) = \frac{k(\mathbf{x}, \boldsymbol{\zeta}_t^*)}{\sum_{t'=1}^{t'=T} k(\mathbf{x}, \boldsymbol{\zeta}_{t'}^*)}, \quad k(\mathbf{x}, \boldsymbol{\zeta}_t^*) = e^{-\frac{\|\mathbf{x}-\boldsymbol{\zeta}_t^*\|^2}{2\sigma^2}} \tag{9.4}$$

Furthermore, each contributing potential element here is a summation of two components: $\phi_t(\mathbf{x}) = \phi_t^{\perp}(\mathbf{x}) + \phi_t^0$, whereby the component $\phi_t^{\perp} : \mathbb{R}^m \mapsto \mathbb{R}_+$ is a strictly convex function with a global minima at ζ_t^* and $\phi_t^0 \in \mathbb{R}_+$ is a bias term. As a consequence of the aforementioned decomposition, the gradient of the overall warped potential in (9.4) can also be decomposed as,

$$\nabla\Phi(\mathbf{x}) = \nabla\Phi^{\perp}(\mathbf{x}) + \nabla\Phi^{/\!/}(\mathbf{x}) \tag{9.5}$$

where the gradient component $\nabla\Phi^{\perp}(\mathbf{x})$ causes attractive pull towards the demonstration while the component $\nabla\Phi^{/\!/}(\mathbf{x})$ produces accelerations along the direction of motion of the

demonstration. The motion along the trajectory is direct consequence of monotonically decreasing bias terms ϕ_t^0, such that the negative of the potential gradient aligns with the demonstrated motion. Due to this decomposition, we independently hand-design the function $\phi_t^{\perp}(\mathbf{x})$ as per our desired attractive accelerations towards the demonstration. Next, we learn the bias terms ϕ_t^0 such that the direction of motion governed by the potential at each data-point $\boldsymbol{\zeta}_t^*$ matches the demonstration.

As a first step in this procedure, we choose to go with the following attractive potential element [61],

$$\phi_t^{\perp}(\mathbf{x}) = \frac{1}{\eta}\log\left(e^{\eta\|\mathbf{x}-\boldsymbol{\zeta}_t^*\|} + e^{-\eta\|\mathbf{x}-\boldsymbol{\zeta}_t^*\|}\right), \quad \nabla\phi_t^{\perp}(\mathbf{x}) = s_\eta\big(\|\mathbf{x}-\boldsymbol{\zeta}_t^*\|\big)\frac{\mathbf{x}-\boldsymbol{\zeta}_t^*}{\|\mathbf{x}-\boldsymbol{\zeta}_t^*\|} \tag{9.6}$$

where $\eta > 0$ defines the effective smoothing radius of the function at the origin and $s_\eta(0) = 0$ and $s_\eta(r) \to 1$ as $r \to 0$. For a sufficiently large η, this choice of potential function ensures that the attractive acceleration always has a unit magnitude except in the neighborhood of the center ζ_t^* where it smoothly decreases to zero. A trivial alternative to this function is a quadratic as used by Khansarizadeh et al [112]. However, the gradient of a quadratic function increases linearly with distance which can cause undesirably large accelerations far away from the demonstrations.

Towards the second step in the procedure, we learn the bias terms $\{\phi_t^0\}$. As mentioned before, we require negative natural gradient to match the accelerations from zero velocity. Also, we are only concerned with the direction of motion. Hence the potential learning problem becomes,

$$\begin{aligned}
&\min_{\{\boldsymbol{\phi}_t^0\}} \ \frac{1}{T}\sum_{t=1}^{t=T} \|\hat{\dot{\boldsymbol{\zeta}}}_t^* + \nabla\Phi\big(\boldsymbol{\zeta}_t^*; \{\phi_t^0\}\big)\|^2 + \lambda\|\phi_t^0\|^2 \\
&\text{s.t. } 0 \le \phi_T^0 \le \phi_{t+1}^0 \le \phi_t^0 \quad \forall t = 1, \ldots, (T-1) \\
&\qquad \nabla\Phi(\boldsymbol{\zeta}_T^*) = 0
\end{aligned} \tag{9.7}$$

where $\hat{\dot{\boldsymbol{\zeta}}}_t^* = \frac{\dot{\boldsymbol{\zeta}}_t^*}{\|\dot{\boldsymbol{\zeta}}_t^*\|}$ is the direction of motion (with unit magnitude) and λ is the regularization parameter. Furthermore, the optimization constraints enforce the potential to decrease monotonically along the demonstration with a stationary point at the goal location $\boldsymbol{\zeta}_T$. The aforementioned optimization problem can be solved efficiently with off-the-shelf solvers, e.g. CVX [113].

9.2.3 Learning Riemannian Metric from Multiple Demonstrations

While a single demonstration can imply certain traits of motions, multiple demonstrations can further capture certain properties of the skill that can not be encoded by a single trajectory. For example, if all the demonstrations stay close to each other in a particular region of the state space, it informs that the motion is highly constrained in the region. In such part of the state space, the reproduced trajectories should follow the demonstrations closely so that the skill specifications are satisfied. Therefore, we choose to learn a Riemannian metric $\mathbf{M}(\mathbf{x})$ on the subtask space (manifold) to warp the learned potential function (9.4). The metric expands or contracts the space so that the attractive component is no longer uniform along the trajectories. Figure 9.1c shows an example vector field given by negative gradient of a nominal potential warped by a learned Riemannian metric.

To ensure the stability of the learned system, the Riemannian metric $\mathbf{M}(\mathbf{x})$ needs to be positive definite. Hence, we parameterize the metric by its Cholesky decomposition, $\mathbf{M}(\mathbf{x}) = \mathbf{L}(\mathbf{x})\,\mathbf{L}(\mathbf{x})^\top$, where $\mathbf{L} \in \mathbb{R}^{n\times n}$ is a lower-triangular matrix with positive diagonal entries. Let $\mathbf{l}_d(\mathbf{x}) \in \mathbb{R}^n$ and $\mathbf{l}_o(\mathbf{x}) \in \mathbb{R}^{\frac{1}{2}(n^2-n)}$ denote the vector given by collecting the diagonal and off-diagonal entries of $\mathbf{L}(\mathbf{x})$, respectively. In order for $\mathbf{L}(\mathbf{x})$ to be a Cholesky decomposition of the positive definite matrix $\mathbf{M}(\mathbf{x})$, we require each entries of $\mathbf{l}_d(\mathbf{x})$ to be strictly positive.

We represent the metric matrix as a neural-network with parameter θ. The neural network takes in the coordinate $\mathbf{x} \in \mathbb{R}^n$ and outputs $\mathbf{l}_d(\mathbf{x};\theta)$ and $\mathbf{l}_o(\mathbf{x};\theta)$ (Figure 9.2). To ensure that $\mathbf{l}_d(\mathbf{x};\theta)$ is strictly positive for all $\mathbf{x} \in \mathbb{R}^n$, we take absolute value of the output of the linear layer and add it with a small positive offset $\epsilon > 0$. Hence, lower-triangular matrix $\mathbf{L}(\mathbf{x};\theta)$ is

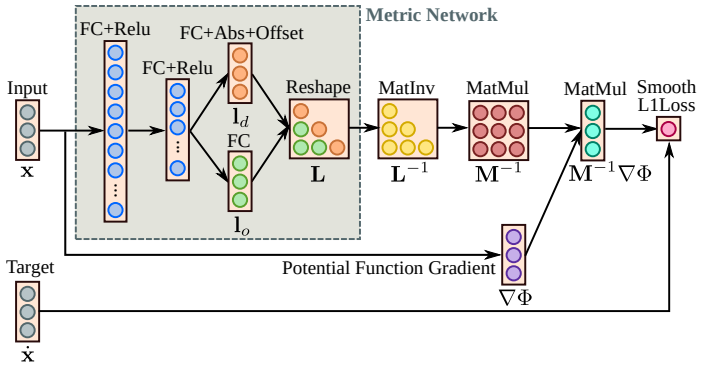


Figure 9.2: The structure of the neural network for metric learning. The first two layers are fully connected layers with Relu activation functions. The diagonal and off-diagonal elements of the lower triangular matrix $\mathbf{L}$ is then predicted through another fully connected layer. In order to ensure that the diagonal elements are strictly positive, the absolute value of the output of the layer is taken and a positive offset is added. Then, the inverse of the metric matrix computed to calculate the loss for training the network.

always invertible and the Riemannian metric $\mathbf{M}(\mathbf{x};\theta)$ is guaranteed to be positive definite. The Riemannian metric learning problem seeks to find the parameters of the neural network such that final natural gradient descent subsystem (negative warped potential gradient) reproduces the demonstrated normalized velocities along the demonstrations,

$$\begin{aligned} \arg\min_{\theta} \quad & \sum_{i=1}^{N}\sum_{t=1}^{T_i} \mathcal{L}\big(-\mathbf{M}(\boldsymbol{\zeta}_{i,t};\theta)^{-1}\nabla\Phi(\boldsymbol{\zeta}_{i,t}),\ \hat{\dot{\boldsymbol{\zeta}}}_{i,t}\big) \\ \text{s.t.} \quad & \mathbf{M}(\boldsymbol{\zeta}_{i,t};\theta) = \mathbf{L}(\boldsymbol{\zeta}_{i,t};\theta)\,\mathbf{L}(\boldsymbol{\zeta}_{i,t};\theta)^{\top}, \end{aligned} \tag{9.8}$$

where $\mathcal{L}(\cdot,\cdot)$ can be any regression loss function, e.g. L2 loss, smooth L1 loss, and their regularized version. It is noteworthy that $\mathbf{M}(\cdot;\theta)$ need not be explicitly computed during training since $\mathbf{M}(\mathbf{x};\theta)^{-1} = \mathbf{L}(\mathbf{x};\theta)^{-\top}\mathbf{L}(\mathbf{x};\theta)^{-1}$, and $\mathbf{L}(\mathbf{x};\theta)^{-1}$ can be efficiently computed through forward-substitution due to the fact that $\mathbf{L}(\mathbf{x};\theta)$ is a lower-triangular matrix. Although the training is done on the first-order system. While executing the learned policy, the curvature term $\boldsymbol{\xi}_{\mathbf{M}}(\mathbf{x},\dot{\mathbf{x}};\theta)$ needs to be computed as well. By definition of the curvature term (9.2), its calculation involves computing the partial derivatives $\frac{\partial M_{ij}(\mathbf{x};\theta)}{\partial x_k}$, which can be computed through back-propagation using standard deep learning frameworks such as PyTorch [102].

Limitations: First, we assume in this work that only the geometric features (i.e. positions and direction of motion/shape) of the demonstrated motions are important for the task. Reproducing the speed profiles of demonstrations would perhaps require learning the entire second-order system in (9.1) with a loss defined on the weighted-combination of policies given by RMPflow. Second, the Riemannian metric can not rotate the attractive potential field by more than 90 degrees. This limitation will manifest itself when demonstrations have large variations in their geometric features.

9.3 Experimental Evaluation

We evaluated the proposed approach on two manipulation tasks, including *door reaching* and *drawer closing*[3]. Both tasks were demonstrated on a 7-DOF Franka Emika robot with configuration space coordinate $\mathbf{q} \in \mathbb{R}^7$. For each skill, a human subject provided 6 demonstrations via kinesthetic teaching, starting from various joint angles of the robot. The demonstrations were recorded in the joint space such that the ith demonstration is defined as $\boldsymbol{\zeta}_i^{\boldsymbol{q}} \coloneqq \{\boldsymbol{\zeta}_{i,t}^{\boldsymbol{q}}\}_{t=0}^{T_i}$, where $\boldsymbol{\zeta}_{i,t}^{\boldsymbol{q}} \in \mathbb{R}^7$.

The desired tasks require coordinated motion of the entire hand of the robot with respect to the target objects. To encode the motion of the entire wrist, we pick $k = 3$ control points on the hand; one defined at the center of the gripper and one each at the tips of the two-fingered gripper. We setup an RMP-tree with root in the joint space of the robot and leaf nodes representing 3-dimensional subtasks, one per control point, under the mappings $\{\psi^{(k)}\}_{k=1}^{3}$. Specifically, a subtask map is defined as a composition $\psi^{(k)} \coloneqq g^{(k)} \circ f^{(k)}$. Here $f^{(k)} : \mathbb{R}^7 \mapsto \mathbb{R}^3$ maps the joint angles to the kth control point position under the robot's forward kinematics. On the other hand, $g^{(k)}(\mathbf{x}) = \mathbf{x} - \mathbf{p}^{(k)}$ defines a translation such that the target location $\mathbf{p}^{(k)} \in \mathbb{R}^3$ coincides with the origin of the subtask space [4]. For each subtask, we independently learn a *human-guided* RMP of the form (9.1) under the transformed subtask space demonstrations $\{\boldsymbol{\zeta}_i^{(k)}\}_{i=1}^{N}$. Appendix A.2 provides additional details on the

[3] Video available at: https://youtu.be/9jvz5fE_1dM.

[4] Target location $\mathbf{p}^{(k)}$ is attached to the target object. This enables the policy to react to object displacement.

learning pipeline and the integrated system.

To execute the skill in a new environment, additional hand-specified joint limit RMPs, and obstacle avoidance RMPs are also added as leaf RMPs to the RMP-tree. These hand-specified RMPs dictate the kinematic and environmental constraints as detailed in [61]. All the policies are resolved in real-time under the learned and hand-specified Riemannian metrics by recursively running RMPflow on the RMP-tree. It should be noted that the kinematic constraints enforced along the RMP-tree ensures the coordination of motion in various subtask spaces.

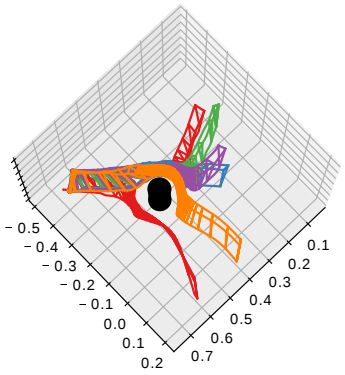
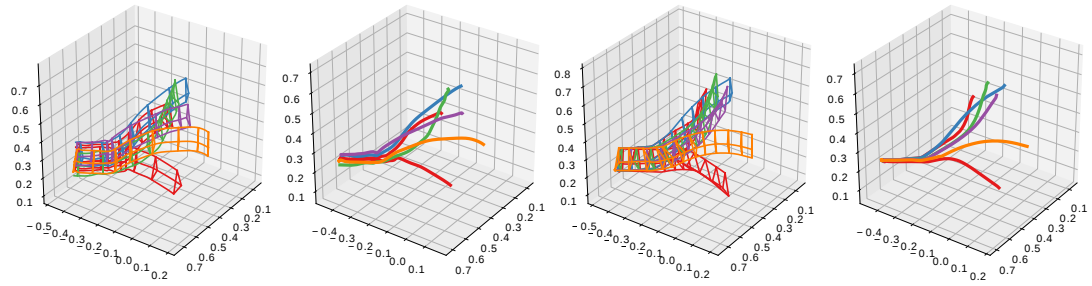


Figure 9.3: Reproductions of the *drawer closing* task with a cylindrical obstacle (in black) in the scene. The positions of the 3 control points on robot hand are shown as vertices of a triangle.

The *drawer closing* task required the robot to reach the handle of a drawer from a given initial position and push it closed. Figure 9.4a–9.4b shows the demonstrations transformed in the 3 aforementioned subtask spaces. It should be noted that the closing motion not only requires a straight line motion by the end-effector after making contact with the drawer handle, but also requires the wrist to align with the face of the handle. Figure 9.4c–9.4d shows the reproductions from the same initial positions as the demonstrations while Figure 9.5 shows an instance of reproduction on the real robot. Among all the 6 trials, the robot is able to successfully reproduce the demonstrations and close the drawer.

The *door reaching* task required the robot to start from inside a cabinet, going around the cabinet door and reaching for the door handle outside the door. The robot is required to stop at a standoff position a small distance away from the drawer handle. This task requires a highly constrained motion that results in the end-effector moving along a C-shaped arc. Furthermore, as before, the entire wrist trajectory is important here since the robot is required to reach the handle at a certain relative angle. Figure 9.6a–9.6b shows the demonstrated trajectories while Figure 9.6c–9.6d shows the reproductions from the same set of initial



(a) (b) (c) (d)

Figure 9.4: Trajectories for the *drawer closing* task. (a), (c): The motion of 3 control points on the robot hand, shown as vertices of a triangle, along the demonstrated and reproduced trajectories respectively. (b), (d): The demonstrated and reproduced end-effector (center of hand) trajectories.

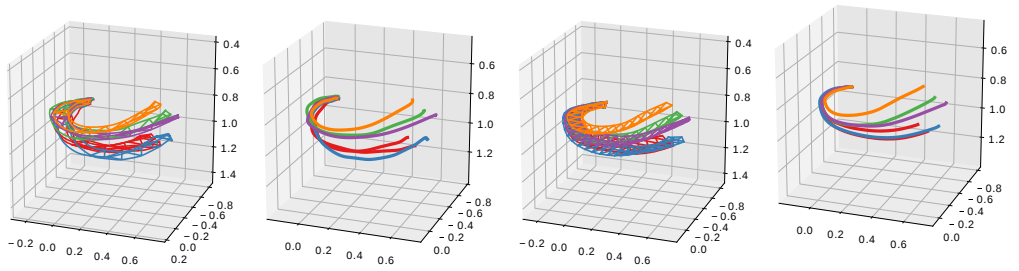
Figure 9.5: The *drawer closing* task. The robot successfully closes the drawer from a new initial configuration.

positions. Figure 9.7 shows snapshots of a task reproduction on the real robot. Once again we notice all the reproductions to successfully achieve the task. Furthermore, to test the reactive behavior of our policy, we displace the door during execution. As noticeable in Figure 9.8, the robot successfully react to the the change in goal location and hence complete the task.

Another important feature our approach inherits from RMPflow is obstacle avoidance. We desire the generated motions to be collision-free in order to be feasible in any new environment. To test the obstacle avoidance behavior, we place a cylindrical obstacle in the environment such that it hinders the robot's motion towards the drawer for the *drawer closing* task. We notice successful completion of the task as the robot is able to go around the obstacles in all 6 trials (Figure 9.3).

9.4 Conclusion

We introduced an approach for learning and reproducing complex tasks, composed of multiple inter-related subtasks. Our approach is capable of learning inherently-stable



(a) (b) (c) (d)

Figure 9.6: Trajectories for the *cabinet door reaching* task. (a), (c): The motion of 3 control points on the robot hand, shown as vertices of a triangle, along the demonstrated and reproduced trajectories respectively. (b), (d): The demonstrated and reproduced end-effector (center of hand) trajectories.

Figure 9.7: The cabinet *door reaching task*. The robot manages to reach the cabinet door handle despite of the new initial configuration and new door configuration.

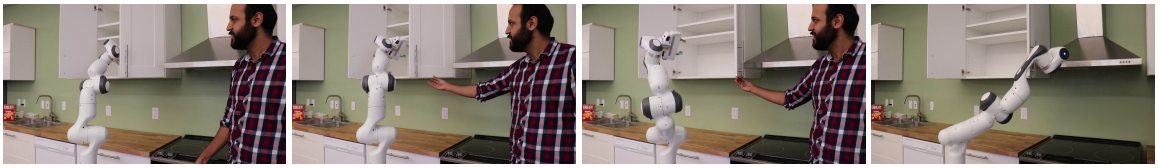


Figure 9.8: Reactivity. The door handle location is displaced during execution and the robot can be seen to adapting to the new target.

reactive polices in these subtask spaces directly from human demonstrations. For task reproduction, our method utilizes RMPflow to carry out policy resolution and generate a stable joint-space policy that enables simultaneous execution of various learned subtasks. Furthermore, the motions generated by the combined policy adhere to the robot's kinematics and environmental constraints. Experimental results demonstrate that the proposed approach can capture the desired behaviors of multiple robot links in order to accomplish constrained manipulation tasks.

CHAPTER 10
CONCLUSION AND FUTURE DIRECTIONS

Robots may often struggle in executing diverse skills in real-world settings, but the structured skill representations presented in this dissertation, aimed at learning robot skills from human demonstrations, bring us one step closer to a world where that is no longer the case. Specifically, this dissertation has shown that:

structure in skill representations, dictated by domain knowledge, can enable sample-efficient learning and generalization of diverse robot skills, capable of generating robot motions that are feasible and often coordinated.

10.1 Summary of contributions

To justify the aforementioned statement, this dissertation has made several contributions, which are divided into three parts.

Part I presented a large-scale benchmarking study (Chapter 3) aimed at highlighting key limitations of prior works in the skill learning community. While this benchmarking effort reported several insights and recommendations, one key finding was that there may not exist a representation that is capable of encoding the sheer diversity of skills out there. In view of this finding, this dissertation presented a family of structured skill representations, each aimed at a subset of skills.

The first set of skill representations, proposed in Part II, included time-dependent methods, namely MCCB and CLAMP. These representations find a new robot trajectory by solving an optimization problem that minimizes a time-accrued cost. In MCCB (Chapter 4), the cost is defined by a weighted combination of costs in multiple differential coordinates, and is learned from demonstrations. On the other hand, CLAMP (Chapters 5–6) views the demonstration-guided cost function as a structured trajectory prior, and employs it in

a probabilistic inference setting, a problem dual to the trajectory optimization problem. The probabilistic inference formulation employed by CLAMP, directly yields configuration space trajectories which are also kinematically and environmentally feasible. MCCB on the other hand, relies on an inverse kinematics solver or motion planner to find configuration commands corresponding to the workspace motions generated by it, which is not only an inefficient process but may also lead to suboptimal behaviors. On the flip side, MCCB allows greater flexibility than CLAMP, since it encodes motions in multiple coordinate systems, instead of relying on a single (i.e. Cartesian) coordinate system.

The second set of structured skill representations, presented in Part III, involved time-invariant representations, i.e. SDSEF and TSDS. Unlike time-dependent methods which optimize an entire trajectory in an offline fashion, time-invariant methods find motions that can instantly react to perturbations. SDSEF (Chapter 7) learns a time-invariant dynamical system from demonstrations, rolling out which reproduces a desired skill. Stability guarantees built into the structure, parameterized by a diffeomorphism, enables robot motions that are bounded and converge to a goal. SDSEF is shown to be capable of representing a variety of task space or workspace motions. To find corresponding configuration space motions, a tracking control law can be utilized. Furthermore, TSDS (Chapter 8–9) extends SDSEF by defining a dynamical system made up of a weighted combination of several task space dynamical systems, each taking a form similar to SDSEF, while preserving stability. The combined dynamical system is formulated directly in the configuration space, thus yielding kinematically feasible motions. By associating each dynamical system to a particular robot body part, TSDS enables learning coordinated motions of several robot body parts.

In summary, this dissertation has presented a broad spectrum of structured skill representations and associated learning frameworks. The representations are explicitly parameterized by either the geometric properties of motions, a prior probability distribution over trajectories, or a dynamical system. While some methods find new motions that are optimal in terms of learned or pre-specified costs (or a combination thereof), others find reactive

motions suitable for dynamic environment. A wide-ranging benchmarking study alongside sufficient qualitative and quantitative evaluations are provided to supplement the theoretical contributions made in this dissertation.

10.2 Future directions

While the methods presented in this dissertation have shown promising results in several different tasks and environments, they are not without limitations. We summarize some of these limitations here and provide possible future directions aimed towards mitigating these limitations.

First, this dissertation assumes that the low-levels skills are independent of the high-level task. There may exist tasks, where the task decomposition structure evolves over time. In such situations, it is pertinent to devise methods that together reasons about the high-level task alongside its constituent skills. In fact, there exist some recent developments in this regime [114, 72, 115, 116]. Future extensions of this dissertation can take inspiration from the existing works to devise combined task and skill learning approaches.

Second, while the existence of structure in representation is beneficial in terms of sample-efficiency, it also introduces model biases. We exploit the model bias in each representation to make it suitable for certain skills. This however puts additional burden on the end-user, requiring him to select an appropriate learning approach beforehand. A possible solution to this problem is to employ an ensemble learning [117] method that learns to find the representation, or a combination of representations, suitable for a given skill. Another potential solution is learn a single unstructured representation, capable of encoding all sorts of skills. Perhaps the biggest hurdle in employing such unstructured learning methods is lack of large amounts of data. Methods for collecting large amounts of data from humans, either in simulation [118], or via multiple physical interactions [119, 120] can be useful in this regard. However, it is unclear if hard constraints like feasibility and coordination can be satisfied by unstructured methods even when large amounts of data is present.

Third, our learning formulations assume that all the human demonstrations are equally good in executing a given skill. However, as we have found in our benchmarking efforts, demonstration quality is highly correlated with demonstrator experience level. Therefore, it is pertinent to differentiate *good* demonstrations from *bad* ones. Perhaps one way of achieving this is by additionally collecting ratings of the demonstrations. The ratings can be given by the demonstrator himself, or can be gathered from other sources (e.g. the AMT users in our benchmarking work). Incorporating these ratings in the learning routine can perhaps help biasing the skill model towards good demonstrations.

Lastly, all our works are limited by the human demonstrator's knowledge about the desired skill. In other words, human demonstrations are assumed to be *optimal*. In the presence of sparse rewards, it is possible to go beyond suboptimal demonstrations using reinforcement learning [121, 122]. Concretely, one may design binary reward signals associated with achieving the goals of a task. For instance, a task aimed at closing the lid of a box, can have a reward function which equals one when the box is successfully closed, but stays zero otherwise. Such a reward signal can be employed in a reinforcement learning setting to explore the state-space beyond demonstrations, and thus refine the skill model. However, ensuring safety of the robot and its environment while carrying out any such exploratory behavior however is critical, which is an active area of research itself [123].

Appendices

APPENDIX A
ADDITIONAL BACKGROUND ON RMPFLOW AND EXPERIMENTAL DETAILS SUPPLEMENTING LEARNING EXPERIMENTS

A.1 RMPflow

In this appendix, we provide a brief introduction of RMPflow [61], the computational framework for policy generation with RMPs. We refer the readers to [61] for detailed introduction and theoretical analysis of RMPflow.

Two components that form the basis of RMPflow include: *1*) RMP-tree: a directed tree encoding the structure of the task map, and *2*) RMP-algebra: a set of operations to propagate information across the RMP-tree. An RMP-tree is a directed tree initiating at the root node, branching out and culminating at the leaf nodes, with edges connecting the parent-child node pairs. Specifically, each node $\mathtt{v}$ along the RMP-tree is made up of a state $(\mathbf{x}, \dot{\mathbf{x}})$ on a manifold along with an associated RMP $(\mathbf{a}_{\mathtt{v}}, \mathbf{M}_{\mathtt{v}})^{\mathcal{M}}$. Each edge $\mathtt{e}$ in the RMP-tree corresponds to a smooth map $\psi_{\mathtt{e}}$ from the given parent node manifold to the child node manifold. The root node in the RMP-tree, $\mathtt{r}$, is associated with the state $(\mathbf{q}, \dot{\mathbf{q}})$ in the configuration space $\mathcal{C}$ and its policy $(\mathbf{a}_{\mathtt{r}}, \mathbf{M}_{\mathtt{r}})^{\mathcal{C}}$. Let K be the number of leaf nodes in the RMP-tree. The leaf nodes $\{\mathtt{l}_k\}_{k=1}^K$ are associated with subtask policies $\{(\mathbf{a}_{\mathtt{l}_k}, \mathbf{M}_{\mathtt{l}_k})^{\mathcal{T}_{\mathtt{l}_k}}\}_{k=1}^K$.

The subtask policies along the RMP-tree are combined using RMP-algebra. To illustrate how RMP-algebra operates, consider a node $\mathtt{u}$ in the RMP-tree with N child nodes $\{\mathtt{v}_j\}_{j=1}^N$ (see Figure A.1). Let $\{\mathtt{e}_j\}_{j=1}^N$ denote the edge between the parent node $\mathtt{u}$ and the child nodes $\{\mathtt{v}_j\}_{j=1}^N$. RMP-algebra consists of three operators:

(*i*) `pushforward` propagates the state of a node in the RMP-tree to update the states of its child nodes. Let $(\mathbf{x}, \dot{\mathbf{x}})$ and $\{\mathbf{y}_j, \dot{\mathbf{y}}_j\}_{j=1}^N$ be the state associated with the parent node and the child nodes, respectively. The state of its jth child node $\mathtt{v}_j$ is computed

as $(\mathbf{y}_j, \dot{\mathbf{y}}_j) = (\psi_{\mathtt{e}_j}(\mathbf{x}), \mathbf{J}_{\mathtt{e}_j}(\mathbf{x})\,\dot{\mathbf{x}})$, where $\psi_{\mathtt{e}_j}$ is the smooth map associated with the edge $\mathtt{e}_j$ and $\mathbf{J}_{\mathtt{e}_j} = \partial_{\mathbf{x}}\psi_{\mathtt{e}_j}$ is the Jacobian matrix.

(*ii*) `pullback` propagates the natural form of RMPs $\{[\mathbf{f}_{\mathtt{v}_j}, \mathbf{M}_{\mathtt{v}_j}]^{\mathcal{N}_j}\}_{j=1}^{N}$ from the child nodes to the parent node. The RMP associated with the parent node $[\mathbf{f}_{\mathtt{u}}, \mathbf{M}_{\mathtt{u}}]^{\mathcal{M}}$ is computed as,

$$\mathbf{f}_{\mathtt{u}} = \sum_{j=1}^{N} \mathbf{J}_{\mathtt{e}_j}^{\top}(\mathbf{f}_{\mathtt{v}_j} - \mathbf{M}_{\mathtt{v}_j}\dot{\mathbf{J}}_{\mathtt{e}_j}\dot{\mathbf{x}}), \qquad \mathbf{M}_{\mathtt{u}} = \sum_{j=1}^{N} \mathbf{J}_{\mathtt{e}_j}^{\top}\mathbf{M}_{\mathtt{v}_j}\mathbf{J}_{\mathtt{e}_j}. \tag{A.1}$$

The natural form of RMPs are used since they more efficient to combine.

(*iii*) `resolve` maps an RMP from its natural form $[\mathbf{f}_{\mathtt{u}}, \mathbf{M}_{\mathtt{u}}]^{\mathcal{M}}$ to its canonical form $(\mathbf{a}_{\mathtt{u}}, \mathbf{M}_{\mathtt{u}})^{\mathcal{M}}$ with $\mathbf{a}_{\mathtt{u}} = \mathbf{M}_{\mathtt{u}}^{\dagger}\,\mathbf{f}_{\mathtt{u}}$, where $\dagger$ denotes Moore-Penrose inverse.

Given configurations state $(\mathbf{q}(t), \dot{\mathbf{q}}(t))$ at time t, RMPflow computes the global policy $\pi(\mathbf{q}(t), \dot{\mathbf{q}}(t)) = \mathbf{a}_{\mathtt{r}}(\mathbf{q}(t), \dot{\mathbf{q}}(t))$ through the following procedure. The `pushforward` operator is first recursively applied to the RMP-tree to propagate the state associated with each node in the RMP-tree. Then, the subtask policies $\{(\mathbf{f}_{\mathtt{l}_k}, \mathbf{M}_{\mathtt{l}_k})\}_{k=1}^{K}$ are evaluated by the leaf nodes and combined recursively along the RMP-tree by the `pullback` operator. the `resolve` operator is finally applied on the root node to compute the acceleration policy $\mathbf{a}_{\mathtt{r}}(\mathbf{q}(t), \dot{\mathbf{q}}(t))$.

A.2 Details of the Experiments

Here we provide some details of the experiments, including the learning pipeline and the integrated perception and control system.

A.2.1 The Learning Pipeline

Data Collection For both the drawer closing experiment and the cabinet door reaching experiment, we collect 6 human demonstrations starting from different initial configurations

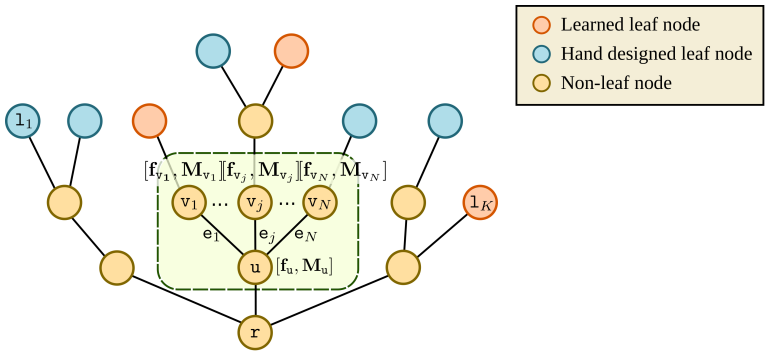


Figure A.1: An example RMP-tree. The root of RMP-tree is associated with the configuration space while the leaf nodes are associated with subtasks. The policies for the subtasks can be either hand-designed or learned. The RMP-algebra propagates information along the tree. The green block contains a segment of the RMP-tree to illustrate the RMP-algebra.

of the robot through kinesthetic teaching. The initial configuration of the environment, i.e., the cabinet door and the drawer, respectively, remains fixed across different demonstrations. For the drawer closing experiment, in particular, the initial configuration of the drawer is always fixed across the data collection phase and the testing phase. In the cabinet door reaching experiment, however, the initial configuration of the cabinet door is recorded by the perception system (see Appendix A.2.2). In the testing phase, we vary the initial configuration of the cabinet door across different trials. The demonstrations are recorded in the joint space.

Data Preprocessing We define the subtask spaces in the reference frame of the object, i.e., the cabinet door handle and the center of the drawer front, respectively. In particular, we consider 3 control points on the wrist of the robot making up 3 subtask spaces of $\mathbb{R}^3$. We also translate the origin of the subtask spaces such that all demonstrations converge to the origin in each subtask spaces. Hence, given the demonstrations in the joint space, we transfer the demonstrations into the subtask spaces using the forward kinematics of the robot composited with a rigid body transformation.

To preprocess the demonstrated trajectories, we first remove the static segments in the trajectories. The trajectories are then smoothed with a Savitzky-Golay filter. We resample the trajectories with 200 sample points evenly spaced in time using cubic interpolation. The

velocity at the sample points are computed through finite-difference. Since we assume that the magnitude of the velocities are not important to the skill, we rescale the velocity at each sample point so that the trajectory has unit velocity for each sample point.

Metric Learning The number of nodes in the two fully connected layers are 128 and 64, respectively. The neural network is trained by the Adam optimizer [103] with a learning rate of 0.005 and weight decay of 3×10^{-5}. Since the size of the training set is relatively small, we use batch update and terminate training after 1000 updates. The average training time for metric learning on a subtask space is 15.76s. The reported training time is an average over 10 runs on a CPU machine with an Intel Core i7 processor.

A.2.2 The Integrated System

Drawer Closing Experiment No perception systems are used for the drawer closing experiment. The initial configuration of the drawer is fixed throughout data collection and testing. The robot is considered moving in an empty space except for the trial where a simulated cylinder-shaped obstacle is added to the environment.

Cabinet Door Reaching Experiment In the cabinet door reaching experiment, the configuration of the robot and the environment, e.g., cabinet door, is tracked by an overhead RGB-D kinect camera using DART [124]. The perception system updates the state of the robot and the environment at 30Hz.

Hand-crafted RMPs In addition to the learned RMPs, obstacle avoidance RMPs and joint limit RMPs detailed in [61] are also active during the testing phase to avoid collision with obstacles and exceeding the joint limits.

Control Frequency The learned RMPs operate at 50Hz during execution while the other hand-crafted RMPs operate at 1000Hz.

REFERENCES

[1] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration", *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.

[2] H. Ravichandar, A. S. Polydoros, S. Chernova, and A. Billard, "Recent advances in robot learning from demonstration", *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, 2020.

[3] S. Chernova and A. L. Thomaz, "Robot learning from human teachers", *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 8, no. 3, pp. 1–121, 2014.

[4] P. Pastor, M. Kalakrishnan, L. Righetti, and S. Schaal, "Towards associative skill memories", in *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, IEEE, 2012, pp. 309–315.

[5] S. Niekum, S. Osentoski, G. Konidaris, S. Chitta, B. Marthi, and A. G. Barto, "Learning grounded finite-state representations from unstructured demonstrations", *The International Journal of Robotics Research*, vol. 34, no. 2, pp. 131–157, 2015.

[6] S. Krishnan, A. Garg, S. Patil, C. Lea, G. Hager, P. Abbeel, and K. Goldberg, "Transition state clustering: Unsupervised surgical trajectory segmentation for robot learning", *The International Journal of Robotics Research*, vol. 36, no. 13-14, pp. 1595–1618, 2017.

[7] S. H. Lee, I. H. Suh, S. Calinon, and R. Johansson, "Autonomous framework for segmenting robot trajectories of manipulation task", *Autonomous robots*, vol. 38, no. 2, pp. 107–141, 2015.

[8] S. Calinon, F. Guenter, and A. Billard, "On learning, representing, and generalizing a task in a humanoid robot", *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 37, no. 2, pp. 286–298, 2007.

[9] S. M. Khansari-Zadeh and A. Billard, "Learning stable nonlinear dynamical systems with Gaussian mixture models", *IEEE Transactions on Robotics*, vol. 27, no. 5, pp. 943–957, 2011.

[10] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Learning attractor landscapes for learning motor primitives", in *Advances in neural information processing systems*, 2003, pp. 1547–1554.

[11] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: Learning attractor models for motor behaviors", *Neural computation*, vol. 25, no. 2, pp. 328–373, 2013.

[12] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann, "Probabilistic movement primitives", in *Advances in neural information processing systems*, 2013, pp. 2616–2624.

[13] D. Koert, G. Maeda, R. Lioutikov, G. Neumann, and J. Peters, "Demonstration based trajectory optimization for generalizable robot motions", in *IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, IEEE, 2016, pp. 515–522.

[14] G. Ye and R. Alterovitz, "Demonstration-guided motion planning", in *Int. Symp. on Robotics Research (ISRR)*, vol. 5, 2011.

[15] S. M. Khansari-Zadeh and A. Billard, "Learning control lyapunov function to ensure stability of dynamical system-based robot reaching motions", *Robotics and Autonomous Systems*, vol. 62, no. 6, pp. 752–765, 2014.

[16] H. C. Ravichandar and A. Dani, "Learning position and orientation dynamics from demonstrations via contraction analysis", *Autonomous Robots*, pp. 1–16, 2018.

[17] N. Perrin and P. Schlehuber-Caissier, "Fast diffeomorphic matching to learn globally asymptotically stable nonlinear dynamical systems", *Systems & Control Letters*, vol. 96, pp. 51–59, 2016.

[18] K. Neumann and J. J. Steil, "Learning robot motions with stable dynamical systems under diffeomorphic transformations", *Robotics and Autonomous Systems*, vol. 70, pp. 1–15, 2015.

[19] M. A. Rana, D. Chen, J. Williams, V. Chu, S. R. Ahmadzadeh, and S. Chernova, "Benchmark for skill learning from demonstration: Impact of user experience, task complexity, and start configuration on performance", in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[20] H. Ravichandar, S. R. Ahmadzadeh, M. A. Rana, and S. Chernova, "Skill acquisition via automated multi-coordinate cost balancing", in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[21] M. A. Rana, M. Mukadam, S. R. Ahmadzadeh, S. Chernova, and B. Boots, "Skill generalization via inference-based planning", in *RSS Workshop on Mathematical Models, Algorithms, and Human-Robot Interaction*, 2017.

[22] M. A. Rana, M. Mukadam, S. R. Ahmadzadeh, S. Chernova, and B. Boots, "Learning generalizable robot skills from demonstrations in cluttered environments", in *2018*

IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2018, pp. 4655–4660.

[23] M. A. Rana, A. Li, D. Fox, B. Boots, F. Ramos, and N. Ratliff, "Euclideanizing flows: Diffeomorphic reduction for learning stable dynamical systems", 2020.

[24] A. Rahimi and B. Recht, "Random features for large-scale kernel machines", in *Advances in neural information processing systems*, 2008, pp. 1177–1184.

[25] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Movement imitation with nonlinear dynamical systems in humanoid robots", in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, IEEE, vol. 2, 2002, pp. 1398–1403.

[26] D.-H. Park, H. Hoffmann, P. Pastor, and S. Schaal, "Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields", in *IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, IEEE, 2008, pp. 91–98.

[27] H. Hoffmann, P. Pastor, D.-H. Park, and S. Schaal, "Biologically-inspired dynamical systems for movement generation: Automatic real-time goal adaptation and obstacle avoidance", in *2009 IEEE International Conference on Robotics and Automation*, IEEE, 2009, pp. 2587–2592.

[28] J. Kober and J. Peters, "Learning motor primitives for robotics", in *2009 IEEE International Conference on Robotics and Automation*, IEEE, 2009, pp. 2112–2118.

[29] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal, "Learning and generalization of motor skills by learning from demonstration", in *IEEE Int. Conf. on Robotics and Automation(ICRA)*, IEEE, 2009, pp. 763–768.

[30] A. D. Dragan, K. Muelling, J. A. Bagnell, and S. S. Srinivasa, "Movement primitives via optimization", in *IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2015, pp. 2339–2346.

[31] Y. Meirovitch, D. Bennequin, and T. Flash, "Geometrical invariance and smoothness maximization for task-space movement generation", *IEEE Transactions on Robotics*, vol. 32, no. 4, pp. 837–853, 2016.

[32] T. Nierhoff, S. Hirche, and Y. Nakamura, "Spatial adaption of robot trajectories based on laplacian trajectory editing", *Autonomous Robots*, vol. 40, no. 1, pp. 159–173, 2016.

[33] M. Vochten, T. De Laet, and J. De Schutter, "Generalizing demonstrated motion trajectories using coordinate-free shape descriptors", *Robotics and Autonomous Systems*, vol. 122, p. 103 291, 2019.

[34] D. P. Losey and M. K. O'Malley, "Trajectory deformations from physical human–robot interaction", *IEEE Transactions on Robotics*, vol. 34, no. 1, pp. 126–138, 2017.

[35] S. Calinon, Z. Li, T. Alizadeh, N. G. Tsagarakis, and D. G. Caldwell, "Statistical dynamical systems for skills acquisition in humanoids", in *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, IEEE, 2012, pp. 323–329.

[36] S. Calinon, P. Kormushev, and D. G. Caldwell, "Compliant skills acquisition and multi-optima policy search with em-based reinforcement learning", *Robotics and Autonomous Systems*, vol. 61, no. 4, pp. 369–379, 2013.

[37] S. Calinon, I. Sardellitti, and D. G. Caldwell, "Learning-based control strategy for safe human-robot interaction exploiting task and robot redundancies", in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Citeseer, 2010, pp. 249–254.

[38] P. Kormushev, S. Calinon, and D. G. Caldwell, "Approaches for learning human-like motor skills which require variable stiffness during execution", in *IEEE Intl Conf. on Humanoid Robots (Humanoids), Workshop on Humanoid Robots Learning from Human Interaction*, 2010.

[39] K. Kronander and A. Billard, "Online learning of varying stiffness through physical human-robot interaction", in *2012 IEEE International Conference on Robotics and Automation*, Ieee, 2012, pp. 1842–1849.

[40] P. Englert, A. Paraschos, M. P. Deisenroth, and J. Peters, "Probabilistic model-based imitation learning", *Adaptive Behavior*, vol. 21, no. 5, pp. 388–403, 2013.

[41] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, "Using probabilistic movement primitives in robotics", *Autonomous Robots*, vol. 42, no. 3, pp. 529–551, 2018.

[42] Y. Huang, L. Rozo, J. Silvério, and D. G. Caldwell, "Kernelized movement primitives", *The International Journal of Robotics Research*, vol. 38, no. 7, pp. 833–852, 2019.

[43] A. Y. Ng, S. J. Russell, *et al.*, "Algorithms for inverse reinforcement learning.", in *Icml*, vol. 1, 2000, p. 2.

[44] S. Calinon, "A tutorial on task-parameterized movement learning and retrieval", *Intelligent Service Robotics*, vol. 9, no. 1, pp. 1–29, 2016.

[45] S. Calinon, D. Bruno, and D. G. Caldwell, "A task-parameterized probabilistic model with minimal intervention control", in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2014, pp. 3339–3344.

[46] T. Osa, A. M. G. Esfahani, R. Stolkin, R. Lioutikov, J. Peters, and G. Neumann, "Guiding trajectory optimization by demonstrated distributions", *Robotics and Automation Letters*, vol. 2, no. 2, pp. 819–826, 2017.

[47] A. Ghalamzan, C. Paxton, G. D. Hager, and L. Bascetta, "An incremental approach to learning generalizable robot tasks from human demonstration", in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, IEEE, 2015, pp. 5616–5621.

[48] M. Kalakrishnan, P. Pastor, L. Righetti, and S. Schaal, "Learning objective functions for manipulation", in *2013 IEEE International Conference on Robotics and Automation*, IEEE, 2013, pp. 1331–1336.

[49] A. Bajcsy, D. P. Losey, M. K. O'Malley, and A. D. Dragan, "Learning robot objectives from physical human interaction", *Proceedings of Machine Learning Research*, vol. 78, pp. 217–226, 2017.

[50] S. R. Ahmadzadeh and S. Chernova, "Trajectory-based skill learning using generalized cylinders", *Frontiers in Robotics and AI*, vol. 5, pp. 1–18, 2018.

[51] N. M. Patrikalakis and T. Maekawa, *Shape interrogation for computer aided design and manufacturing*. Springer Science & Business Media, 2009.

[52] H. K. Khalil, "Nonlinear systems", 2002.

[53] K. M. Lynch and F. C. Park, *Modern Robotics*. Cambridge University Press, 2017.

[54] M. Mukadam, J. Dong, X. Yan, F. Dellaert, and B. Boots, "Continuous-time Gaussian process motion planning via probabilistic inference.", *The International Journal of Robotics Research (IJRR)*, 2018.

[55] B. Schölkopf, C. J. Burges, A. J. Smola, *et al.*, *Advances in kernel methods: support vector learning*. MIT press, 1999.

[56] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *et al.*, "Gradient-based learning applied to document recognition", *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[57] W. Grathwohl, R. T. Chen, J. Betterncourt, I. Sutskever, and D. Duvenaud, "Ffjord: Free-form continuous dynamics for scalable reversible generative models", *arXiv preprint arXiv:1810.01367*, 2018.

[58] E. G. Tabak and C. V. Turner, "A family of nonparametric density estimation algorithms", *Communications on Pure and Applied Mathematics*, vol. 66, no. 2, pp. 145–164, 2013.

[59] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real nvp", *arXiv preprint arXiv:1605.08803*, 2016.

[60] K. Neumann, A. Lemme, and J. J. Steil, "Neural learning of stable dynamical systems based on data-driven lyapunov candidates", in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013, pp. 1216–1222.

[61] C.-A. Cheng, M. Mukadam, J. Issac, S. Birchfield, D. Fox, B. Boots, and N. Ratliff, "RMPflow: A computational graph for automatic motion policy generation", *Proceedings of the 13th Annual Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2018.

[62] S. Calinon and A. Billard, "Statistical learning by imitation of competing constraints in joint space and task space", *Advanced Robotics*, vol. 23, no. 15, pp. 2059–2076, 2009.

[63] A. Paraschos, R. Lioutikov, J. Peters, and G. Neumann, "Probabilistic prioritization of movement primitives", *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2294–2301, 2017.

[64] J. Silvério, S. Calinon, L. Rozo, and D. G. Caldwell, "Learning task priorities from demonstrations", *IEEE Transactions on Robotics*, vol. 35, no. 1, pp. 78–94, 2019.

[65] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, J. Peters, *et al.*, "An algorithmic perspective on imitation learning", *Foundations and Trends® in Robotics*, vol. 7, no. 1-2, pp. 1–179, 2018.

[66] O. Kroemer, S. Niekum, and G. Konidaris, "A review of robot learning for manipulation: Challenges, representations, and algorithms", *arXiv preprint arXiv:1907.03146*, 2019.

[67] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods", *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, p. 21, 2017.

[68] A. Lemme, Y. Meirovitch, S. M. Khansari-Zadeh, T. Flash, A. Billard, and J. J. Steil, "Open-source benchmarking for learned reaching motion generation in robotics", *Paladyn, Journal of Behavioral Robotics*, vol. 6, no. 1, 2015.

[69] S. R. Ahmadzadeh, M. A. Rana, and S. Chernova, "Generalized cylinders for learning, reproduction, generalization, and refinement of robot skills.", in *Robotics: Science and Systems*, 2017.

[70] M. Buhrmester, T. Kwang, and S. D. Gosling, "Amazon's mechanical turk: A new source of inexpensive, yet high-quality, data?", *Perspectives on psychological science*, vol. 6, no. 1, pp. 3–5, 2011.

[71] M. Müller, "Dynamic time warping", *Information retrieval for music and motion*, pp. 69–84, 2007.

[72] G. Konidaris, S. Kuindersma, R. Grupen, and A. Barto, "Robot learning from demonstration by constructing skill trees", *The International Journal of Robotics Research*, vol. 31, no. 3, pp. 360–375, 2012.

[73] O. Kroemer, C. Daniel, G. Neumann, H. Van Hoof, and J. Peters, "Towards learning hierarchical skills for multi-phase manipulation tasks", in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2015, pp. 1503–1510.

[74] F. Meier, E. Theodorou, F. Stulp, and S. Schaal, "Movement segmentation using a primitive library", in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, IEEE, 2011, pp. 3407–3412.

[75] R. P. Adams and D. J. MacKay, "Bayesian online changepoint detection", *arXiv preprint arXiv:0710.3742*, 2007.

[76] S. M. Khansari-Zadeh, *Lasa handwriting dataset*, https://bitbucket.org/khansari/lasahandwritingdataset.

[77] H. C. Ravichandar, I. Salehi, and A. P. Dani, "Learning partially contracting dynamical systems from demonstrations.", in *Conference on Robot Learning*, 2017, pp. 369–378.

[78] S. Levine and V. Koltun, "Learning complex neural network policies with trajectory optimization", in *International Conference on Machine Learning*, 2014, pp. 829–837.

[79] J. Umlauft and S. Hirche, "Learning stable stochastic nonlinear dynamical systems", in *International Conference on Machine Learning*, 2017, pp. 3502–3510.

[80] M. A. Rana, M. Mukadam, S. R. Ahmadzadeh, S. Chernova, and B. Boots, "Towards robust skill generalization: Unifying learning from demonstration and motion planning", in *Conference on Robot Learning*, 2017, pp. 109–118.

[81] Y. Lipman, O. Sorkine, D. Cohen-Or, D. Levin, C. Rossi, and H.-P. Seidel, "Differential coordinates for interactive mesh editing", in *Shape Modeling Applications*, IEEE, 2004, pp. 181–190.

[82] D. A. Cohn, Z. Ghahramani, and M. I. Jordan, "Active learning with statistical models", *Journal of artificial intelligence research*, vol. 4, pp. 129–145, 1996.

[83] M. M. Fréchet, "Sur quelques points du calcul fonctionnel", *Rendiconti del Circolo Matematico di Palermo (1884-1940)*, vol. 22, no. 1, pp. 1–72, 1906.

[84] M. Mukadam, J. Dong, X. Yan, F. Dellaert, and B. Boots, "Continuous-time Gaussian process motion planning via probabilistic inference", *arXiv preprint arXiv:1707.07383*, 2017.

[85] F. Dellaert, "Factor graphs and gtsam: A hands-on introduction", Georgia Institute of Technology, Tech. Rep., 2012.

[86] C. E. Rasmussen and C. K. Williams, *Gaussian processes for machine learning*. MIT press Cambridge, 2006, vol. 1.

[87] S. Anderson, T. D. Barfoot, C. H. Tong, and S. Särkkä, "Batch nonlinear continuous-time trajectory estimation as exactly sparse Gaussian process regression", *Autonomous Robots*, vol. 39, no. 3, pp. 221–238, 2015.

[88] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm", *IEEE Transactions on information theory*, vol. 47, no. 2, pp. 498–519, 2001.

[89] J. Dong, M. Mukadam, F. Dellaert, and B. Boots, "Motion planning as probabilistic inference using Gaussian processes and factor graphs", in *Proceedings of Robotics: Science and Systems (RSS-2016)*, 2016.

[90] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space", *Intelligent Data Analysis*, vol. 11, no. 5, pp. 561–580, 2007.

[91] S. M. Khansari-Zadeh and A. Billard, "A dynamical system approach to realtime obstacle avoidance", *Autonomous Robots*, vol. 32, no. 4, pp. 433–454, 2012.

[92] A. Rai, F. Meier, A. Ijspeert, and S. Schaal, "Learning coupling terms for obstacle avoidance", in *IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, IEEE, 2014, pp. 512–518.

[93] A. Gams, M. Denisa, and A. Ude, "Learning of parametric coupling terms for robot-environment interaction", in *IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, IEEE, 2015, pp. 304–309.

[94] T. Minka, "Bayesian linear regression", Citeseer, Tech. Rep., 2000.

[95] S. Schaal, "Is imitation learning the route to humanoid robots?", *Trends in cognitive sciences*, vol. 3, no. 6, pp. 233–242, 1999.

[96] S. M. Khansari-Zadeh, *Lasa handwriting dataset*, https://bitbucket.org/khansari/lasahandwritingdataset.

[97] M. A. Rana, A. Li, H. Ravichandar, M. Mukadam, S. Chernova, D. Fox, B. Boots, and N. Ratliff, "Learning reactive motion policies in multiple task spaces from human demonstrations", in *Conference on Robot Learning*, 2020, pp. 1457–1468.

[98] J. P. Hespanha, *Linear systems theory*. Princeton university press, 2018.

[99] J. M. Lee, *Riemannian manifolds: an introduction to curvature*. Springer Science & Business Media, 2006, vol. 176.

[100] S.-I. Amari, "Natural gradient works efficiently in learning", *Neural computation*, vol. 10, no. 2, pp. 251–276, 1998.

[101] T. Flash and N. Hogan, "The coordination of arm movements: An experimentally confirmed mathematical model", *Journal of neuroscience*, vol. 5, no. 7, pp. 1688–1703, 1985.

[102] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch", in *NIPS Autodiff Workshop*, 2017.

[103] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", *arXiv preprint arXiv:1412.6980*, 2014.

[104] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series.", in *KDD workshop*, Seattle, WA, vol. 10, 1994, pp. 359–370.

[105] H. chaandar Ravichandar and A. Dani, "Learning position and orientation dynamics from demonstrations via contraction analysis", *Autonomous Robots*, vol. 43, no. 4, pp. 897–912, 2019.

[106] M. Mukadam, C.-A. Cheng, D. Fox, B. Boots, and N. Ratliff, "Riemannian motion policy fusion through learnable lyapunov function reshaping", in *Conference on Robot Learning*, 2020, pp. 204–219.

[107] N. D. Ratliff, J. Issac, D. Kappler, S. Birchfield, and D. Fox, "Riemannian motion policies", *arXiv preprint arXiv:1801.02854*, 2018.

[108] J. J. Craig, *Introduction to robotics: mechanics and control, 3/E*. Pearson Education India, 2009.

[109] M. A. Alvarez, L. Rosasco, and N. D. Lawrence, "Kernels for vector-valued functions: A review", *arXiv preprint arXiv:1106.6251*, 2011.

[110] V. Sindhwani, M. H. Quang, and A. C. Lozano, "Scalable matrix-valued kernel learning for high-dimensional nonlinear multivariate regression and granger causality", *arXiv preprint arXiv:1210.4792*, 2012.

[111] F. Bullo and A. D. Lewis, *Geometric control of mechanical systems: modeling, analysis, and design for simple mechanical control systems*. Springer New York, 2005.

[112] S. M. Khansari-Zadeh and O. Khatib, "Learning potential functions from human demonstrations with encapsulated dynamic and compliant behaviors", *Autonomous Robots*, vol. 41, no. 1, pp. 45–69, 2017.

[113] M. Grant, S. Boyd, and Y. Ye, *CVX: Matlab software for disciplined convex programming*, 2009.

[114] D. Kappler, P. Pastor, M. Kalakrishnan, M. Wüthrich, and S. Schaal, "Data-driven online decision making for autonomous manipulation.", in *Robotics: Science and Systems*, 2015.

[115] S. Niekum, S. Osentoski, G. Konidaris, and A. G. Barto, "Learning and generalization of complex tasks from unstructured demonstrations", in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2012, pp. 5239–5246.

[116] R. A. Gutierrez, V. Chu, A. L. Thomaz, and S. Niekum, "Incremental task modification via corrective demonstrations", in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 1126–1133.

[117] C. Zhang and Y. Ma, *Ensemble machine learning: methods and applications*. Springer, 2012.

[118] A. Mandlekar, Y. Zhu, A. Garg, J. Booher, M. Spero, A. Tung, J. Gao, J. Emmons, A. Gupta, E. Orbay, *et al.*, "Roboturk: A crowdsourcing platform for robotic skill learning through imitation", *arXiv preprint arXiv:1811.02790*, 2018.

[119] A. Bajcsy, D. P. Losey, M. K. O'Malley, and A. D. Dragan, "Learning from physical human corrections, one feature at a time", in *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, 2018, pp. 141–149.

[120] S. Calinon and A. Billard, "Active teaching in robot programming by demonstration", in *RO-MAN 2007-The 16th IEEE International Symposium on Robot and Human Interactive Communication*, IEEE, 2007, pp. 702–707.

[121] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations", *arXiv preprint arXiv:1709.10087*, 2017.

[122] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients", *Neural networks*, vol. 21, no. 4, pp. 682–697, 2008.

[123] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning", *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[124] T. Schmidt, R. A. Newcombe, and D. Fox, "Dart: Dense articulated real-time tracking.", in *Robotics: Science and Systems*, Berkeley, CA, vol. 2, 2014.